IMPARIAMO IL PASCAL SUL NOSTRO COMPUTER

con i programmi compilatori per i computer
IBM PC, IBM compatibili, OLIVETTI M 10 - M 20 - M 21 - M 24,
HP 150 e ZX SPECTRUM

di JEREMY RUSTON

IMPARIAMO IL PASCAL SUL NOSTRO COMPUTER

con i programmi compilatori per i computer IBM PC, IBM compatibili, OLIVETTI M 10 - M 20 - M 21 - M 24, HP 150 e ZX SPECTRUM

di JEREMY RUSTON

Traduzione di Rosella Boaron

Via dei Lavoratori, 124
CINISELLO BALSAMO (MI)

Questo libro si rivolge a chi desidera conoscere il Pascal
ed apprenderne l'uso in modo semplice e piano:
e' quindi adatto anche a chi e' alle prime armi
nel campo dell'informatica.

Il libro e' corredato da due programmi per tradurre
le istruzioni Pascal in Basic: questo consente al lettore
di provare direttamente programmi in Pascal sul suo
personal computer senza dover affrontare la spesa
di un vero compilatore Pascal.

Il primo traduttore e' scritto in Basic Microsoft,
quindi e' adatto a quasi tutti i personal computers
come IBM, OLIVETTI, H.P., ...

Il secondo e' scritto in Basic Sinclair per home computers
della serie ZX SPECTRUM.

INDICE

APPENDICI

CAPITOLO PRIMO

INTRODUZIONE AL PASCAL

La scelta di un linguaggio di programmazione e' spesso fonte di indecisione : una volta che si e' scelto il linguaggio (ad esempio il Pascal), un ulteriore problema e' come imparare il limguaggio adottato. Questo libro intende per l'appunto essere di utilita' per due scopi : vi potra' aiutare a decidere se il Pascal e' un linguaggio adatto ai vostri scopi, e, supposto che decidiate per il si', vi insegnera' a programmare in Pascal.

Per un uso efficace del libro dovete avere un calcolatore su cui giri o lo pseudo-compilatore Pascal riportato alla fine di questo libro, o un compilatore Pascal di tipo commerciale. Nel primo caso si avra' la limitazione di dover usare un sottoinsieme limitato dei comandi.

Il libro e' quindi diviso in due parti: la prima parte si riferisce ad entrambi i gruppi di lettori, mentre la seconda si riferisce a quelle caratteristiche del Pascal non presenti nello pseudo-compilatore, quindi puo' essere utile solo a chi possiede un "vero" compilatore Pascal . Quando arriverete alla seconda parte, sarete gia' in grado di decidere se per voi e' preferibile procurarvi una versione commerciale di Pascal per il vostro calcolatore se non ne avete gia' una. Quindi, se trovate che non vi piace programmare in Pascal, non avrete speso dei soldi per un compilatore Pascal solo per provarlo.

Volendo utilizzare questo libro con il mio compilatore (vedi appendice) dovete prima leggerne la descrizione: nell' appendice e' spiegato quale dei compilatori allegati dovete usare e come questi lavorano. Nella prima parte del libro ogni commento che si riferisce esclusivamente al mio compilatore e' racchiuso fra parentesi quadre.

Se usate dei compilatori commerciali, e' difficile dare indicazioni precise su come usarli, poiche' questi operano in modo diverso l'uno dall'altro: il manuale che accompagna ciascun compilatore potra' comunque darvi le necessarie informazioni. Un problema dei compilatori commerciali e' che gli autori tendono a estendere il Pascal quando lo realizzano. Queste estensioni si presentano sotto forma di nuovi comandi e "migliori" modi di fare le cose: nella maggior parte dei casi queste estensioni non interferiranno con il Pascal standard su cui si basa questo libro; alcune estensioni sono abbastanza comuni: queste sono menzionate nel libro in relazione ai relativi argomenti.
La sequenza con cui normalmente si scrive e si esegue un programma Pascal e' pressappoco la seguente :

1) Il programma deve essere battuto sul calcolatore . I compilatori Pascal molto raramente forniscono un mezzo per fare

cio'; spesso dovete comprare un ulteriore programma chiamato "Editore" (in inglese "Editor") per questo scopo. Un Editor e' un programma di tipo generale che consente di creare testi dalla tastiera (possono essere indifferentemente lettere a vostra zia o programmi Pascal), correggerli ed infine salvarli su disco. Le funzioni per effettuare correzioni sono normalmente piuttosto sofisticate: ad esempio permettono di spostare interi blocchi di testo o di ricercare specifiche parole in una frazione di secondo. Un tipico Editor (per calcolatori che usano il sistema operativo CP/M) e' il WordMaster.

2) Dovete dire al compilatore di compilare il programma: cio' spesso si fa lasciando l' ambiente dell' Editor e chiamando il compilatore direttamente da tastiera: il compilatore allora prende il vostro programma Pascal da disco e lo converte in un codice interno che viene anch' esso salvato su disco.

3) Il programma compilato puo' quindi essere eseguito battendo qualche altro comando da tastiera.

Cio' che non e' immediatamente ovvio da quanto detto sopra e' che se fate un errore nel programma Pascal dovete ritornare all' Editor per correggere l' errore e ripetere di nuovo l' intera procedura; questo fa perdere molto tempo in confronto allo scrivere e al correggere programmi in Basic. L'unico modo per aggirare il problema e' essere sicuri che i vostri programmi non contengano errori!

Il Pascal e' stato fatto in questo modo perche' nel 1969/70 i calcolatori erano cosi' costosi che l'unico modo ragionevole di scrivere i programmi era quello di scriverli e verificarli sulla carta prima di introdurli nel calcolatore. Che il Pascal abbia potuto sopravvivere malgrado questo laborioso sistema di introduzione dei programmi e' una testimonianza della sua eleganza e utilita' .

Tutti i programmi in Pascal, siano semplici o complessi, hanno lo stesso formato generale:

```
PROGRAM ESEMPIO(INPUT,OUTPUT);
-
-
-
BEGIN
-
-
-
END.
```

Il primo gruppo di trattini rappresenta la sezione di inizializzazione del programma, che dice al calcolatore cio' che gli serve sapere per cominciare effettivamente a far girare il

programma, le cui istruzioni compaiono nel secondo gruppo di trattini.

Le parole stampate in **grassetto** sono chiamate "parole riservate". Sono riservate per scopi speciali e l' utente non puo' utilizzarle per nessun altro scopo. La maggior parte delle altre parole che appaiono nei programmi Pascal sono chiamate "identificatori" .

La linea:

PROGRAM ESEMPIO(INPUT,OUTPUT) ;

Si chiama "intestazione del programma" : fornisce al calcolatore varie informazioni preliminari che gli sono necessarie prima che possa iniziare il programma (ad esempio il nome del programma stesso). Nel caso riportato sopra, il nome del programma e' "ESEMPIO". Potete dare ai vostri programmi in Pascal qualunque nome vi piaccia a condizione che questo nome:

a) non sia una parola riservata (queste variano da un compilatore all' altro, quindi controllate il vostro manuale per avere una lista completa delle parole da evitare).

b) inizi con una lettera

c) i caratteri seguenti siano una combinazione di lettere o numeri.

E' una buona idea chiamare un programma con lo stesso nome che userete come nome del file quando lo salverete sul floppy disk.

Il nome del programma e' un esempio di identificatore, anche se al calcolatore non importa come chiamate un programma; e' sensato usare nomi che danno un' idea di cosa fa il programma. Questi sono alcuni esempi:

PROGRAM CALCOLATASSE(INPUT,OUTPUT) ;
PROGRAM SCACCHI(INPUT,OUTPUT) ;
PROGRAM TROVARADICEQUADRATA(INPUT,OUTPUT);

Poiche' uno spazio bianco non e' una lettera dell'alfabeto il nome del programma non deve contenere caratteri di spaziatura; questo puo' dar luogo a nomi di programmi come quelli sopra riportati, scomodi da leggere a prima vista. Per questa ragione molte versioni del Pascal consentono di utilizzare il carattere di sottolineatura (_) negli identificatori. Questo consente di scrivere identificatori complessi senza sacrificare la leggibilita'. L'ultimo esempio sopra riportato potrebbe essere riscritto come:

PROGRAM TROVA_RADICE_QUADRATA(INPUT,OUTPUT) ;

Riportiamo ora qualche esempio di intestazione illegale a causa di errori nel formato dell' identificatore:

PROGRAM 12TABELLATEMPI(INPUT,OUTPUT);
(comincia con un numero)
PROGRAM TOM&JERRY(INPUT,OUTPUT);
(contiene un carattere non ammesso)
PROGRAM LED ZEPPELIN(INPUT,OUTPUT);
(contiene uno spazio)

In teoria non vi sono limitazioni nella lunghezza degli identificatori, ma quasi tutte le versioni del Pascal considerano significativi solo i primi otto caratteri di ciascun identificatore. Quindi l'identificatore PROGRAMMA_PAGHE e' trattato allo stesso modo di PROGRAMMA_DISEGNI .

Nomi lumghi sono di solito scomodi da battere e da ricordare, quindi e' preferibile usare versioni abbreviate dei nomi. Per esempio e' piu' frequente chiamare un programma CONT_GEN piuttosto che PROGRAMMA_DI_CONTABILITA'_GENERALE.

Dopo il nome viene una coppia di parentesi. Per le finalita'di questo libro queste parentesi dovrebbero sempre contenere le parole "INPUT,OUTPUT".

La fine dell'intestazione del programma e' indicata con un punto e virgola . Il Pascal usa il punto e virgola nello stesso modo con cui noi usiamo il punto in italiano, cioe' per indicare la fine di un comando o di una frase. Per l'uso del punto e virgola nei programmi Pascal vi sono regole molto precise che verranno spiegate in seguito.

Ritornando al nostro esempio iniziale ricorderete che c'erano due blocchi di trattini: il primo viene chiamato "sezione dichiarativa", il secondo "corpo del programma".

La sezione dichiarativa da' al calcolatore alcune informazioni preliminari necessarie per poter eseguire il programma: per esempio bisogna indicare in questa sezione i nomi di tutte le variabili di un programma.

Il corpo del programma contiene le istruzioni che costituiscono il programma stesso.

Possiamo ora esaminare un programma Pascal, che potreste voler introdurre nel vostro calcolatore. [Nel mio compilatore dovete omettere sempre la prima linea del programma poiche' esso non richiede l'intestazione].

```
PROGRAM SALUTA(INPUT,OUTPUT);
BEGIN
 WRITE ('HELLO')
```

Si noti che questo programma non include la parte dichiarativa.

Il corpo del programma e' fatto di una sola istruzione: WRITE ('HELLO') .

Tutte le istruzioni nel corpo di un programma Pascal contengono una coppia di parentesi in cui sono contenuti i dati su cui l'istruzione dovra' operare. In qualche raro caso l'istruzione non deve operare su dati e quindi le parentesi vengono omesse.

Nel caso dell'istruzione WRITE, il contenuto delle parentesi e' stampato sullo schermo televisivo. Il modo con cui questo avviene dipende da questi dati. Tutti i caratteri chiusi fra apici (') saranno semplicemente stampati sullo schermo cosi' come sono: cosi' nell'esempio suddetto l'istruzione WRITE scrive la parola "HELLO" nello schermo. E' importante notare che gli apici stessi non sono stampati, poiche' sono usati per delimitare la stringa di caratteri e non fanno parte di essa. Volendo includere il simbolo di apice in un comando di WRITE bisogna quindi usare le virgolette (").

Avrete notato che la parola WRITE non e' una parola riservata (non e'in grassetto), diversamente dalla PRINT del Basic a cui equivale. La ragione di cio' verra' chiarita nell'ultima parte del libro, ma per il momento tutto cio' che dovete sapere e' che WRITE e' un identificatore standard.

Una volta che voi abbiate il programma fatto e funzionante potreste volerlo modificare per stampare testi differenti.

Il seguente programma amplia alcuni dei concetti pecedenti:

```
PROGRAM CIRCONFERENZA(INPUT,OUTPUT)

 CONST PI=3.1415926;
 VAR RAGGIO,CERCHIO:REAL;

BEGIN { blocco del programma principale }

 WRITE ('INTRODURRE IL RAGGIO DEL CERCHIO');
 READLN (RAGGIO)
 CERCHIO:=RAGGIO*2*PI;
 WRITELN ('CIRCONFERENZA = ',CERCHIO)

END.
```

Le linee vuote hanno lo scopo di rendere il programma piu' leggibile evidenziando le suddivisioni fra sezioni differenti. Possono essere omesse quando introducete il programma, poiche' non influiscono sul suo comportamento, ma non e' ragionevole sacrificare la leggibilita' dei vostri programmi per risparmiare qualche battuta.

La sezione dichiarativa del programma precedente e' fatta di due parti: la dichiarazione CONST e la dichiarazione VAR . La sezione dichiarativa "CONST" crea un identificatore chiamato PI con il "valore" 3.1415926; questo identificatore di costante puo' quindi essere usato in sostituzione di qualunque numero nei successivi calcoli. La dichiarazione di costante e' formata dalla parola CONST seguita da un identificatore, dal segno di uguale e dal valore dell'identificatore. Il termine di ciascuna dichiarazione e' indicato da un altro punto e virgola. Se questo punto e virgola e' seguito da un altro identificatore, viene creata un' altra costante, altrimenti e' seguito da una parola riservata (VAR in questo caso), su cui si opera come richiesto. Cio' sembra un po' complicato ma vedrete che ha un senso quando lo userete.

La sezione VAR ha una sintassi analoga, ma si possono dichiarare allo stesso tempo piu'identificatori separandoli con virgole. La maggior parte degli altri programmi che vedrete illustra cio'.
Gli identificatori dichiarati con la sezione VAR sono variabili anziche' costanti. Le variabili sono identificatori che possono avere valori ad essi assegnati durante il corso di un programma. Esse possono anche essere trattate come le costanti in quanto possono apparire nei calcoli. Le memorie delle calcolatrici tascabili si comportano in un certo senso come delle variabili.

La differenza tra variabili e costanti consiste nel fatto che il valore assegnato a una costante non puo' cambiare nel programma mentre le variabili possono essere modificate quando si vuole.

Vi sono alcune buone ragioni per usare le costanti nei programmi invece che normali variabili, quando cio' e' possibile. Se scrivete un programma per fare qualche calcolo complesso e quindi lo stampate , volete essere sicuri che i risultati non vengano stampati sulle perforazioni che separano i fogli di carta della stampante. Potreste quindi fare in modo che il programma stampi un numero di linee bianche quando la stampante e' vicina al fondo del foglio per spostarvi all'inizio del foglio successivo. In questo caso e' bene dichiarare una costante chiamandola "PAGESIZE" (dimensione pagina) con un valore uguale a 66 (che e' il normale numero di linee per pagina), in modo che se dovesse cambiare il formato della carta sia sufficiente modificare solo la sezione di dichiarazione delle costanti senza andare attraverso tutto il programma cercando i numeri corrispondenti alle linee per pagina. In parole povere le costanti possono rendere i programmi piu' facilmente manutenibili. Un altro motivo puo' essere una miglior leggibilita': piuttosto che veder spuntar fuori nel vostro programma un 3.1415926, e' piu' sensato usare una costante chiamandola PI.

BEGIN indica l'inizio del programma principale.

La prima linea del programma principale e' un comando di WRITE.

Nell' esempio questa istruzione avverte l'utente che nel programma deve essere introdotto il raggio del cerchio.

Il comando READLN nella riga successiva riceve un numero dalla tastiera e lo memorizza come variabile RAGGIO. Da questo punto in poi la variabile RAGGIO puo' essere trattata esattamente come un numero e quando apparira' in un calcolo l'elaboratore utilizzera' il valore immesso. L'unico modo per cambiare il numero memorizzato come RAGGIO e' di usare l'identificatore in un altro comando READ, o assegnare un valore differente ad esso in un comando di assegnazione.

Il comando successivo e' un comando di assegnazione: questo e' indicato dalla presenza della coppia di simboli := . Quando il calcolatore incontra questo comando elabora il valore alla destra di questo simbolo e lo assegna alla variabile alla sinistra del simbolo stesso.

Cio' che si trova alla destra del simbolo := e' chiamato espressione. Una espressione nel Pascal puo' essere una costante, una variabile, o una combinazione di entrambe collegate per mezzo di operatori. Gli operatori sono i segni di addizione, sottrazione, moltiplicazione, divisione. Nell'esempio e' usato solo l' operatore di moltiplicazione (*) . Il risultato dell'espressione e' la circonferenza di un cerchio di raggio RAGGIO. Quindi, secondo il concetto sopra espresso di usare identificatori con nomi descrittivi, il risultato e' memorizzato nella variabile CERCHIO.

Vi sono alcune differenze fra l'aritmetica del Pascal e la matematica delle calcolatrici tascabili. Innanzitutto il Pascal usa dei simboli particolari per alcune operazioni; si usa l' asterisco per indicare la moltiplicazione (come detto sopra) e si usa la barra (/) per la divisione; i segni + e - sono usati nel modo normale. La seconda differenza e' nel modo in cui le espressioni sono effettivamente calcolate. In alcune calcolatrici battendo 2+2*8 i calcoli vengono eseguiti nell'ordine con cui sono battuti gli operatori e quindi si ha come risultato 32. Questa espressione nel Pascal da' come risultato 18 : questo perche'il Pascal esegue le moltiplicazioni e le divisioni prima di fare le addizioni e le sottrazioni, cio' secondo le corrette regole matematiche. Quindi in questo esempio il Pascal calcola 2*8, e quindi somma 2 . Volendo modificare l'ordine con cui l'espressione e' calcolata si possono utilizzare coppie di parentesi : l' espressione (2+2)*8 da' in Pascal come risultato 32 .

(Diversamente dal Basic il valore di una variabile prima che compaia in una READLN o in un comando di assegnazione e' imprevedibile: quindi tutte le variabili devono essere attentamente inizializzate prima che appaiano in un comando di READ o alla destra di un comando di assegnazione).

Il comando di assegnazione in Pascal e' un po' differente da quello Basic. In Basic solitamente si scrive "A=23",omettendo la parola LET. Potreste meravigliarvi che in Pascal si usi il simbolo " : " prima del segno " = " . La verita' e' che chi ha inventato il Pascal era un purista ed un ottimista. Era un purista, poiche' aveva notato che il segno " = " usato in questo contesto non era corretto: una linea come "T=T+1" , come apparirebbe in BASIC, potrebbe generare confusione nei principianti, perche' T non e',e non potra' mai essere, uguale a T+1. Percio'egli intendeva aiutare i principianti indicando con segni diversi i due concetti diversi di "uguale" e di "metti nella variabile" . Cio' era molto ottimistico da parte sua, perche' non aveva capito che quasi tutti imparano a programmare in calcolatori che funzionano in Basic.

L' ultima linea nel programma principale e' un nuovo tipo di comando di output (uscita), il comando WRITELN. La differenza tra WRITELN e WRITE e' che WRITELN salta alla riga successiva dopo che ha effettuato l'operazione di scrittura.

Il comando

WRITELN;

fa saltare alla riga successiva per il successivo comando di output, cioe' se il precedente comando di output era un WRITE ci si sposta alla riga successiva, se era un WRITELN si stampa una riga bianca: puo' quindi essere usato per aumentare la spaziatura fra le righe per migliorare la leggibilita'.

[Nel mio compilatore dovete usare " WRITELN (' '); " che semplicemente stampa uno spazio e quindi passa alla riga successiva. Per questo motivo, quando e' il caso' usero' sempre questa forma nella prima parte del libro].

Il comando WRITELN nel programma dimostra che e' anche possibile stampare numeri . Quando trova che c'e' come output un identificatore, il comando non stampa la parola (ad esempio RAGGIO), ma tira fuori il valore corrente della variabile RAGGIO e lo stampa. Quando vi sono due dati da stampare essi devono essere separati da virgole nel comando WRITELN.

La parola riservata " END. " indica la fine del comando WRITELN, quindi non e' necessario il punto e virgola. Indica anche la fine del programma, come abbiamo visto prima.

Le parole racchiuse in parentesi graffe dopo la parola riservata BEGIN sono commenti. Come tali sono ignorati dal calcolatore, ma possono essere utili al programmatore per chiarire il programma ad altri che lo leggano, o come promemoria per se stesso. I programmi piu' complicati in questo libro sono molto ben commentati, poiche' e' importante che li capiate. [Il mio compilatore non gestisce commenti] . Se il vostro calcolatore

non ha parentesi graffe sulla tastiera, potete usare " (* " per iniziare un commento e " *) " chiusa per terminarlo. Pero'
{ INIZIO PROGRAMMA *) non e' un commento valido.
Siamo ora in grado di allargare la descrizione del formato generale di un programma Pascal; il nuovo programma e' del tipo:

```
PROGRAM NOMEPROG(INPUT,OUTPUT)
 CONST
 -
 -
 -
 VAR
 -
 -
 -
 BEGIN
 -
 -
 -
END.
```

Se scorrete questo libro potete notare che tutti i programmi comprendono alcune o tutte queste parole caratteristiche: si noti che le sezioni WAR e CONST che appaiono nello schema in qualche programma potrebbero mancare, ma BEGIN e END, sono sempre necessari.

Ho inserito dei caratteri di spaziatura aggiuntivi all' inizio di alcune linee del programma: questi spazi bianchi fanno rientrare la scrittura per meglio evidenziare ciascuna sezione. Man mano che il formato generale dei programmi diventera' piu' complesso, vi renderete conto dell'importanza delle rientranze. Alcvni tipi di Basic effettuano delle rientranze nei comandi FOR-NEXT con la stessa finalita'.

Il seguente programma illustra altri aspetti del Pascal:

```
PROGRAM PAGHE(INPUT,OUTPUT)

CONST
 PAGA=3.5;
VAR
 ORE:REAL;
 GIORNI:REAL;

BEGIN

 WRITE ('DARE ORE LAVORATE PER GIORNO E NUMERO GIORNI');
 READLN (ORE,GIORNI);
 WRITE ('PAGA TOTALE  = 'PAGA*ORE*GIORNI);

END.
```

Questo esempio ha il difetto di essere ovvio, ma illustra alcuni punti importanti. Nella sezione WAR e' mostrato il concetto di avere piu' di una dichiarazione. Il comando READLN in questo caso ha due argomenti: come potete vedere se si deve introdurre piu' di una variabile i diversi nomi delle variabili sono separati da virgole. Lo stesso vale per quasi tutti i comandi Pascal che possono avere piu' di un argomento.

CAPITOLO SECONDO

SEMPLICI TIPI DI DATI E COMANDO IF

La parola REAL e' stata usata in una sezione dichiarativa VAR nell' ultimo capitolo senza spiegazioni.

Quattro differenti parole possono occupare il posto di REAL nella sezione VAR. Ciascuna parola conferisce differenti caratteristiche alla variabile cosi' creata. Ciascuno dei differenti "tipi" e' qui spiegato con le caratteristiche ad esso associate.

Il tipo REAL

Una variabile dichiarata di tipo REAL e'definita come un numero reale, cioe'come un numero che puo' includere un punto decimale.

Numeri come 2.5 o 4.o sono reali. I numeri reali possono essere sommati, sottratti, moltiplicati e divisi secondo le regole dell' ultimo capitolo.

I numeri reali sono i piu' simili al tipo di numeri a cui siamo abituati nella vita di tutti i giorni. Comunque il Pascal e' stato progettato soprattutto per essere usato come un linguaggio scientifico ed educativo, quindi usa un formato per certi numeri reali che sembrera' estraneo alla maggior parte dei lettori. Questo formato e' conosciuto come notazione "scientifica" o "esponenziale".

I numeri scritti nell'annotazione esponenziale sono composti di due parti: la mantissa e l'esponente. Queste due parti sono separate dalla lettera ' E '. Percio' la forma di un numero esponenziale e' <mantissa> E <esponente>. La mantissa e' un normale numero compreso fra 1 e 10. Se il numero ha meno di 10 cifre, e' riempito con zeri (cioe' 2.3 diventera' 2.300000000). Quando un numero in formato esponenziale e' digitato in un calcolatore questi zeri possono essere omessi.

La mantissa e' seguita dalla lettera E e quindi dall' esponente. L' esponente e' la potenza di 10 per cui deve essere moltiplicata la mantissa per dare il numero in questione. Per esempio 2.300000000E4 e' 2.3 per 10 elevato alla quarta , cioe' 23000.

Voi non siete costretti a introdurre i numeri in questo formato, ma il Pascal in mancanza di diverse indicazioni stampa i numeri in questo modo: questo non e' sempre molto comodo. Tuttavia e' data la possibilita' di controllare il formato di stampa a seconda delle necessita' dell' utente: un numero reale in un comando di WRITE o WRITELN puo' essere seguito da due numeri (o

variabili) che danno la posizione e la larghezza del campo entro cui stampare il numero con le sue cifre decimali. Per esempio WRITELN (PI:20:6) stampa PI (supposto che il suo valore sia stato precedentemente definito) appoggiato a destra (cioe' senza spazi in coda) in modo tale che la distanza dell'ultima cifra del numero dal lato sinistro del foglio o dello schermo sia di 20 caratteri. La stampa e' limitata a 6 cifre decimali .

Questa possibilita' e' utile nelle applicazioni in cui un utente voglia usare il calcolatore senza troppe sofisticazioni; altrimenti il calcolatore stamperebbe una soluzione esageratamente precisa di un problema (ad esempio un tasso di conversione dollaro/lira di 1.573042379E3) .

[Questa possibilita' non e' prevista dal mio compilatore. In generale e' piu' sicuro omettere simboli di formattazione in un comando di WRITE].

L'aritmetica fra numeri reali non e' sempre esatta; supponendo che il calcolatore visualizzi solo 10 cifre, dara' come risultato di 123456789+0.000000000001 il numero 123456789, che e' solo approssimativamente giusto.

Il tipo INTEGER

Le variabili intere sono come le variabili reali nel senso che rappresentano dei numeri; tuttavia le variabili intere possono solo rappresentare numeri che non contengano un punto decimale. Quindi ogni variabile intera puo' essere rappresentata come una variabile reale contenente solo zeri dopo il punto. Questi esempi danno un'idea piu' chiara delle differenze:

NUMERI REALI

2346.34574568+E89
-23.4
-3.0
4.2345
-2.235

NUMERI INTERI

32146
-23
23
643

Poiche' tutti gli interi sono esprimibili come reali si potrebbe essere perplessi sul fatto che il Pascal preveda anche gli interi: le ragioni di cio' sono due, velocita' e precisione.

La maggior velocita' che danno gli interi deriva dal fatto che questi possono essere memorizzati internamente al calcolatore in

due parti separate, mentre un numero reale puo' richiedere fino a sei parti: e' piu' veloce per il calcolatre manipolare due parti piuttosto che sei e cio' rende l'aritmetica intera piu' veloce.

Il motivo per cui le variabili reali possono essere imprecise e' dovuto al fatto che certe frazioni non possono essere espresse in forma binaria nello stesso modo in cui 1/3 non puo' essere convertito in un numero decimale senza perdere di precisione.

Nella pratica Vi consiglio di usare numeri interi,quando possibile, in quanto richiedono meno battitura!

Le regole per l'aritmetica intera sono un po' diverse da quelle dell' aritmetica reale. I quattro simboli sono usati nello stesso modo, salvo che l'operatore di divisione (/) da' un risultato reale anziche' intero. Per ottenere un risultato reale in una divisione e' necessario utilizzare gli operatori DIV e MOD: " X DIV Y " da' la parte intera di X diviso per Y e " X MOD Y " da' il resto di X diviso per Y.

Il campo di variabilita' degli interi e' limitato. Il maggior intero che il Pascal puo' gestire e' dato dalla costante MAXINT. Questa e' presente in tutti i compilatori Pascal [eccetto il mio] Il piu' piccolo intero che puo' essere rappresentato e' dato da -MAXINT. Il valore tipico di MAXINT e'32767. Questo campo di variabilita' e' estremamente limitato in confronto alle variabili reali, ma molti programmi non scientifici possono limitarsi a variabili comprese in questo campo(molti compilatori hanno anche un'estensione degli interi standard detta "interi lunghi" (LONG INTEGERS) con un campo di variabilita' maggiore, ma ovviamente vengono trattati in modo piu' lento e richiedono piu' memoria. E' opportuno che ricerchiate nel manuale Pascal del vostro calcolatore maggiori informazioni su questa estensione.

Le variabili intere sono dichiarate nello stesso modo delle variabili reali, salvo che la parola INTEGER e' sostituita dalla parola REAL nella dichiarazione VAR.

Gli interi nei comandi di WRITE possono essere formattati come le variabili REAL, salvo il fatto che non e' possibile specificare il numero delle cifre decimali in stampa, in quanto negli interi non c'e' il punto decimale. E' possibile specificare la larghezza del campo: questo basta a garantire che la stampa sia ordinata e pulita.

Il tipo CHAR

Le variabili di tipo CHAR sono un po' differenti dalle variabili reali ed intere, poiche' non contengono un numero ma una singola lettera o simbolo. Il seguente programma fornisce un esempio di

impiego di variabile CHAR:

```
PROGRAM PROVA(INPUT,OUTPUT);
VAR
 A,B,C:CHAR;
BEGIN
 WRITE ('DARE 3 LETTERE');
 READLN (A,B,C);
 WRITE (C,B,A)
END.
```

Questo programma vi permette di introdurre tre lettere e quindi le stampa in ordine inverso.

Non dovrebbe essere difficile comprendere questo programma. Il comando READLN agisce in modo un po' differente se applicato a variabili di tipo CHAR piuttosto che a variabili numeriche.

READLN con argomenti di tipo CHAR resta in attesa dell' introduzione di tutti i caratteri uno dopo l'altro (senza dare R ETURN) . Percio' nel programma precedente e' possibile introdurre tutte le vostre lettere senza separarle con dei "RETURN". Il mio compilatore comunque sostituisce READLN con INPUT ed usa variabili di stringa al posto di variabili CHAR, quindi il programma precedente richiederebbe dei RETURN tra ciascun carattere. In realta' il comando READLN assomiglia al comando INPUT del Basic solo quando l' argomento del comando e' numerico. Quando l' argomento e' di tipo CHAR, assomiglia piu' ai comandi INKEY$ o GET del Basic.

Cio' potrebbe sembrare uno svantaggio del mio compilatore, ma poiche' le possibilita' di elaborare caratteri nel PASCAL sono molto limitate in confronto al BASIC, non vi sono molti programmi nella prima parte del libro che riguardano le variabili di tipo CHAR .

[La versione del compilatore relativa allo ZX SPECTRUM ammette solo nomi di un unico carattere per le variabili di tipo CHAR ; i programmi in questo libro useranno normalmente nomi piu' lunghi, quindi gli utenti dello SPECTRUM dovranno cambiare il nome di tutte le variabili di tipo CHAR].

Il tipo BOOLEANO

Questo ultimo tipo di variabile e' anche piu' limitato nel numero di valori differenti che puo' assumere rispetto ai tipi descritti sopra. Le variabili reali possono avere molti milioni di valori, le variabili intere alcune migliaia e le variabili di tipo CHAR alcune centinaia; le variabili booleane possono avere solo due valori differenti!

I due valori sono VERO (TRUE) e FALSO (FALSE). Vi sono quattro operatori che si possono applicare alle variabili booleane, AND,

OR, EOR (indicato talora con EXOR) e NOT. I risultati di questi operatori sono indicati nella seguente tabella:

```
TRUE   AND   TRUE   =   TRUE
TRUE   AND   FALSE  =   FALSE
FALSE  AND   TRUE   =   FALSE
FALSE  AND   FALSE  =   FALSE

TRUE   OR    TRUE   =   TRUE
TRUE   OR    FALSE  =   TRUE
FALSE  OR    TRUE   =   TRUE
FALSE  OR    FALSE  =   FALSE
TRUE   EOR   TRUE   =   FALSE
TRUE   EOR   FALSE  =   TRUE
FALSE  EOR   TRUE   =   TRUE
FALSE  EOR   FALSE  =   FALSE

       NOT   TRUE   =   FALSE
       NOT   FALSE  =   TRUE
```

Questo significa che un'espressione come "AFFAMATO AND POVERO" e' valida. Le possibilita' che danno questi tipi di operatori risultera' evidente dopo la descrizione del comando IF.

Le variabili booleane sono dichiarate e assegnate nel modo normale; l' unica eccezione e' che non si devono usare nei comandi di READLN o WRITE.

Ricorderete che la sezione CONST di un programma non specifica il tipo di costanti dichiarate. Poiche' vi sono quattro tipi differenti di variabili, vi potreste stupire che sia necessario dichiarare esplicitamente i tipi di variabili, ma non i tipi di costanti. La ragione e' che il calcolatore puo' sempre determinare il tipo di una costante semplicemente esaminando il suo valore (per esempio costanti del tipo CHAR sono scritte fra due apici). Poiche' una variabile in una sezione VAR non ha un valore, e' necessario dirne prima il tipo al calcolatore.

La dichiarazione CONST vi permette anche di dichiarare costanti di tipo CHAR che hanno piu' di un carattere.

L'ISTRUZIONE " IF "

L'istruzione IF modifica la sequenza di un programma in funzione dell'esito di certi fatti. Vi sono due forme di istruzione IF :

IF condizione **THEN** istruzione

e:

IF condizione **THEN** istruzione 1
ELSE istruzione 2

Nel primo caso l'istruzione dopo la parola *riservata THEN* e' eseguita soltanto se una condizione dopo la parola IF e' calcolata come variabile booleana VERA. La seconda forma opera allo stesso modo salvo che se la condizione e' FALSA, si esegue l'istruzione 2.

Un' espressione booleana puo' essere formata di due espressioni numeriche o CHAR, unite dai segni di " = " (uguale), " <> " (diverso), " < " (minore), " > " (maggiore), " >= " (maggiore o uguale), " <= " (minore o uguale), " > " (maggiore), " < " (minore).

E' possibile unire due o piu' espressioni in un comando IF utilizzando gli operatori AND , OR , EOR . Un altro tipo di espressione booleana e' quella formata da variabili booleane unite da operatori booleani.

Normalmente bisognerebbe usare delle parentesi in queste condizioni per migliorare la leggibilita' del programma, come si vede in alcuni esempi di questo capitolo.

Se volete usare piu' di un' istruzione in ciascuna parte delle strutture, che sopra abbiamo indicato come "istruzione", dovete usare una struttura BEGIN-END per racchiudere le istruzioni.

Gli esempi di programma seguenti illustrano cio' e gli altri aspetti sopra esposti.

```
PROGRAM DECISIONE(INPUT,OUTPUT);
VAR
 A,B:REAL;
BEGIN
 WRITE ('QUAL'E'LA RADICE DI 400 ? ');
 READLN (A);
 IF A=20 THEN WRITELN ('GIUSTO !')
 ELSE WRITELN ('ERRATO');
 WRITE ('QUANTI ANNI HAI ? ');
 READLN (B);
 IF (B>16) AND (A<>20) THEN
  BEGIN
  WRITELN ('NON SEI MOLTO BRAVO');
  WRITELN ('PER I TUOI ',B,' ANNI')
  END
 ELSE
  BEGIN
  WRITELN ('MOLTO BENE PER ',B,' ANNI')
  END
END.
```

[Il mio compilatore non e' molto brillante nel vedere quando termina un'istruzione IF e inizia l'istruzione THEN. Se la condizione termina con un identificatore, l' intera condizione deve essere racchiusa fra parentesi. Quindi si deve scrivere IF (A=B) THEN, invece di IF A=B THEN.

Il mio compilatore vuole anche che si usi la costruzione BEGIN-END nell'istruzione IF anche quando THEN e' seguito da una sola istruzione. Tutti i programmi nella prima parte del libro rispettano questa regola].

Si noti come non ci sia punto e virgola prima dei comandi END.

Se vi fosse dato il programma precedente senza nessun punto e virgola probabilmente avreste serie difficolta' nell' inserire i punti e virgola nella posizione corretta. Vi sono comunque alcune semplici regole per il posizionamento dei punti e virgola.

Tutti i compilatori Pascal operano sui programmi da compilare esaminando un carattere alla volta. Per esempio il mio compilatore richiama in continuazione la routine alla linea 1770 che fornisce il successivo carattere del programma. Essendo questo l'unico contatto che il compilatore ha con il programma, non puo' sapere nulla dell' esistenza delle interruzioni fra le linee se incontra solo un flusso costante di caratteri. Il punto e virgola consente al compilatore di stabilire dov' e' la fine delle istruzioni, senza le limitazioni dell' a capo come nel Basic.

Il vantaggio di questo approccio e' che e' possibile scrivere programmmi dove un'istruzione e' piu' lunga di una singola linea.

Per essere sicuri che i punti e virgola siano posti nella posizione corretta e' necessario definire rigidamente che cosa intendiamo per "programma PASCAL".

Un programma PASCAL consiste di piu' elementi:

1) la parola **PROGRAM**

2) un identificatore (il nome del programma)

3) una parentesi aperta

4) le parole 'INPUT,OUTPUT'

5) una parentesi chiusa

6) un punto e virgola

7) un blocco
8) un punto

Un blocco e':

1) una dichiarazione **CONST** opzionale

2) una dichiarazione **VAR** opzionale

3) la parola **BEGIN**

4) una o piu' istruzioni, separate da punti e virgola

5) la parola **END**

Si noti che le istruzioni sono separate da punti e virgola, non terminano con questi. Questo e' il motivo per cui non dobbiamo porre un punto e virgola prima del comando END finale.

Un'istruzione puo' essere:

1) un' assegnazione

2) il richiamo di una procedura

3) la parola **Begin** seguita da uno o piu' statements separati da punti e virgola e terminanti con la parola **END**.

4) la parola **IF** , un' espressione, la parola **THEN** , un' istruzione, ed eventualmente la parola **ELSE** e un'ulteriore istruzione.

Vi sono piu' tipi di istruzioni che saranno trattati a tempo debito.

La chiamata ad una procedura e' uno dei comandi **WRITE, WRITELN** o **READLN.**

Seguendo attentamente le precedenti spiegazioni e facendo riferimento all' esempio fatto dovreste essere in grado di comprendere la posizione dei punti e virgola.

Si noti che un'istruzione puo' essere "nulla": cio' significa che non contiene caratteri o contiene solo spazi e commenti. [Il mio compilatore non accetta istruzioni nulle] . Essendo previste istruzioni nulle diventa possibile scrivere un programma di questo tipo:

```
PROGRAM VUOTO (INPUT,OUTPUT);

BEGIN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

END.
```

che consiste di una serie di istruzioni nulle. Avrete notato che questo programma, contrariamente alla regola fondamentale, ha un punto e virgola direttamente prima di un'istruzione END !

CAPITOLO TERZO

FUNZIONI STANDARD E ISTRUZIONE FOR

Una funzione e' un processo che in qualche modo cambia un numero: ad esempio la funzione "radice quadrata" cambia un numero nella sua radice quadrata. In Pascal vi sono molte di queste funzioni predefinite e, come potrete vedere, e' facile definirne di nuove.

Le funzioni sono identificate da identificatori predefiniti: ad esempio la parola SQRT significa "square root" (radice quadrata). Le funzioni sono sempre seguite da una coppia di parentesi che racchiudono l'argomento della funzione, analogalmente a quanto si ha nel comando WRITE. Diversamente dal WRITE una funzione puo' comparire solo in espressioni. Cio' poiche' tutte le funzioni danno come risultato un valore e se una funzione fosse richiamata come parte di un programma allo stesso modo dell'istruzione WRITE non ci sarebbe modo di ritrovare il risultato. Alla fine di questo capitolo vi e' qualche semplice esempio di programma contenente delle funzioni.

In una funzione all' interno delle parentesi vi puo' essere qualunque cosa, da un' altra espressione ad un numero.

Vediamo ora una descrizione completa di tutte le funzioni "standard": qualche compilatore ha altre funzioni in piu', ma nessun compilatore dovrebbe averne meno.

ABS da' il valore assoluto del suo argomento, cioe' il valore stesso dell'argomento senza eventuale segno negativo. Quindi il valore assoluto di -3 e' 3, come il valore assoluto di 3 e' ancora 3. ABS puo' avere un argomento di tipo reale o di tipo intero e il risultato e' dello stesso tipo dell'argomento. ABS si usa essenzialmente in programmi matematici e scientifici; i matematici dicono che questa funzione da' il modulo di un numero, ma non si deve confondere questo con l'operatore MOD.

ARCTAN (talora chiamata ATAN o ATN) da' l' arco tangente del suo argomento: se A e' la tangente di PI in radianti, l'istruzione B=ATN(A) pone B uguale a PI. ARCTAN puo' avere un argomento reale o intero, ma il suo risultato e' sempre reale. [Il mio compilatore riconosce solo la forma ATN]

CHR converte un numero in un carattere;ad ogni carattere che il vostro calcolatore puo' stampare corrisponde un codice numerico (spesso derivato dal codice standard ASCII): la funzione CHR vi consente di convertire il codice interno in un

carattere. Deve avere un argomento intero ed il risultato e' di tipo CHAR.

COS da' il coseno del suo argomento che puo' essere di tipo intero o reale; il risultato e' sempre reale. L'argomento deve essere in radianti.

EXP da' la costante "e" elevata alla potenza data dall'argomento di EXP: questa istruzione puo' essere pensata come l' antilogaritmo naturale di un numero. L' argomento puo' essere reale o intero, il risultato e' sempre reale.

LN da' il logaritmo naturale o neperiano del suo argomento. L' argomento deve essere di tipo reale o intero e il risultato e' di tipo reale.

ODD da' il risultato booleano VERO se il suo argomento e' un numero dispari, FALSO in caso contrario. L' argomento deve essere di tipo intero. Altro modo di usare questa funzione e' per sapere se l'argomento e' divisibile per due o no.

ORD e' l' opposto di CHR in quanto converte un carattere nel codice interno del calcolatore. Il suo argomento deve essere di tipo CHAR ed il suo risultato e' un intero.

PRED restituisce l' intero di un' unita' inferiore al suo argomento o il carattere il cui codice e' di un' unita' inferiore a quello dell' argomento. [Il mio compilatore accetta solo àrgomenti interi] . PRED e' un' abbreviazione per "precedente". Nella maggior parte dei calcolatori si ha PRED ('B')='A', ma e' bene evidenziare cio' nella documentazione del vostro programma.

ROUND restituisce l' intero piu' vicino ad un argomento reale.

SIN restituisce il seno del suo argomento (che deve essere in radianti) di tipo reale od intero. Il risultato e' sempre reale.

SQR (da non confondere con l' operatore SQR del Basic) restituisce il quadrato dell' argomento. L'argomento puo' essere reale o intero, il risultato e' sempre dello stesso tipo del'argomento.

SQRT restituisce la radice quadrata dell'argomento; l' argomento puo' essere reale od intero, ma il risultato e' sempre reale.

SUCC e' l'opposto di PRED, cioe' il successivo del suo argomento; anche qui argomento e risultato saranno del tipo CHAR od intero.

TRUNC e' come ROUND, ma invece di arrotondare l'argomento prende il maggior intero piu' piccolo od uguale al numero di partenza. Quindi ROUND semplicemente arrotonda all'intero piu'

vicino, mentre TRUNC elimina la parte del numero reale dopo il punto decimale. Quindi 1.7 sara' "arrotondato" a 2, ma "troncato" a 1.

Il Pascal standard non ha la funzione TAN. E' semplice, comunque, simularla con SIN(X)/COS(X). Come evidenziato sopra, tutte le funzioni trigonometriche operano solo in radianti. Per convertire da gradi in radianti si deve dividere per 180 e moltiplicare per PI; per convertire radianti in gradi dividere per PI e moltiplicare per 180.

[Il mio compilatore riconosce SUCC,PRED,ROUND,SQR,ODD solo se il calcolatore su cui si usa accetta le funzioni definite dall' utente: cio'dipende dal fatto che queste funzioni standard Pascal non hanno equivalenti Basic. Le definizioni di queste funzioni vengono inserite nelle prime cinque linee del programma Basic prodotto.]

* * * * *

Se volete scrivere un programma per stampare tutti gli interi fra uno e dieci, il solo modo che per ora conoscete e' il seguente:

```
PROGRAM NUMERI(INPUT,OUTPUT) ;

BEGIN
   WRITELN  (1) ;
   WRITELN  (2) ;
   WRITELN  (3) ;
   WRITELN  (4) ;
   WRITELN  (5) ;
   WRITELN  (6) ;
   WRITELN  (7) ;
   WRITELN  (8) ;
   WRITELN  (9) ;
   WRITELN  (10)
END
```

Non siate cosi' zelanti da provare anche questo programma: esso naturalmente funziona, ma che cosa fareste se aveste da stampare tutti gli interi fra uno e mille?
Si puo' piu' semplicemente ripetere una parte di programma un dato numero di volte mediante l'istruzione FOR. Essa assume il seguente formato:

```
FOR identificatore:=inizio TO fine
DO istruzione
```

Come con il comando IF, se si vuole piu' di un' istruzione dopo il DO, si deve usare la struttura BEGIN-END.

L' identificatore e' una variabile di tipo intero.

Il comando iterativo FOR dice al calcolatore di eseguire l' istruzione che segue il DO facendo assumere alla variabile tutti i valori compresi fra "inizio" e "fine": quindi, se inizio era 1 e fine era 10, il "loop" (cioe' l' iterazione) viene eseguita 10 volte.

Se per qualche ragione il valore iniziale e' maggiore di quello finale, cioe' la variabile deve essere decrementata ad ogni passo, la parola TO deve essere sostituita con la parola DOWNTO; ovviamente se si usa DOWNTO bisogna assicurarsi che i due valori siano corretti, cioe' il primo maggiore del secondo.

[Il mio compilatore vuole, comunque, la costruzione BEGIN-END]

Il seguente programma stampa tutti i caratteri del vostro calcolatore (nell' ipotesi che esso usi la codifica ASCII).

```
PROGRAM CARATTERI(INPUT,OUTPUT) ;
VAR
   CH:INTEGER;
BEGIN
  FOR CH:=32 TO 126 DO
  WRITE  (CHR(CH))
END
```

In questo esempio si usa solo un' istruzione dopo la parola DO e, quindi, non e' necessario il costrutto BEGIN-END.

[Lo stesso programma scritto per il mio compilatore diventa:

```
VAR CH:INTEGER;

BEGIN
  FOR CH:=32  TO  126  DO
    BEGIN
      WRITE   (CHR(CH))
    END

END.
```

Si noti il posizionamento dei punti e virgola nel programma].

Nel FOR i valori di inizio e fine dell' iterazione possono essere espressioni o numeri.

Adesso che abbiamo introdotto l' istruzione FOR possiamo affrontare esempi di programi meno banali.

Il seguente programma consente di introdurre 10 numeri e stampare la loro media:

```
PROGRAM  MEDIA(INPUT,OUTPUT) ;
```

```
VAR
  NUM,TOT:REAL;
  N:INTEGER;

BEGIN

  TOT:=0

 FOR N:=1 TO 10 DO
 BEGIN

  WRITE ('INTRODURRE NUMERO ',N);
  READLN (NUM) ;
  TOT:=TOT+NUM
 END;

 WRITELN ('');
 WRITELN ('MEDIA = ',TOT/10)

END
```

La rientranza delle istruzioni e l'uso di righe bianche hanno reso questo programma molto piu' facile da leggere di quanto sarebbe stato altrimenti; per chiarire cio' riportiamo lo stesso programma scritto in un modo piu' sciatto:

```
PROGRAM MEDIA(INPUT,OUTPUT);
VAR
NUM,TOT:REAL;
N:INTEGER;
BEGIN
TOT:=0
FOR N:= 1 TO 10 DO
BEGIN
WRITE ('INTRODURRE NUMERO ',N);
READLN (NUM) ;
TOT:=TOT+NUM
END;
WRITELN ('');
WRITELN ('MEDIA = ',TOT/10)
END
```

In un programma scritto in questo modo e' molto piu' difficile vedere le coppie di BEGIN e END corrispondenti, e l' intera struttura del programma e' nebulosa. In programmi piu' lunghi diventa ancora piu' importante l'uso razionale dei caratteri di spaziatura.

Dopo l' esecuzione di un'iterazione con il FOR, la variabile indice (cioe' la variabile che viene dopo la parola FOR) non puo' piu' essere usata senza riassegnarle un valore: questo perche' il suo valore resta indefinito.

Non e' possibile modificare il valore della variabile indice durante l' esecuzione delle iterazioni.

Un problema legato all' uso del FOR e' che non e' possibile uscire dalle iterazioni senza seguire tutta la sequenza di valori previsti per la variabile indice. Normalmente per aggirare questo problema si usa uno speciale indicatore booleano per decidere se le istruzioni nel loop devono essere eseguite:

```
FLAG := TRUE;

FOR LOOP := 10 TO 1000 DO

BEGIN

IF FLAG THEN

BEGIN

-

-

-

IF condizione THEN FLAG := FALSE

END  {IF}

END  {FOR}
```

Il programma eseguira' solo il contenuto del loop se FLAG e' VERO (TRUE); facendo diventare FLAG FALSO quando si vuole uscire dal loop, tutti i rimanenti cicli saranno percorsi senza eseguire le istruzioni.

Questa soluzione e' probabilmente la piu' elegante (l' eleganza e' un' importante proprieta' dei programmi), ma e' ancora preferibile scrivere il programma in modo da evitare di trovarsi in questa situazione.

La variabile indice e' sempre incrementata o decrementata di un' unita'; se si vuole un passo di un altro valore, bisogna essere un po' tortuosi. Per esempio un programma per stampare una lettera si' e una no dell' alfabeto, potrebbe essere scritto in questo modo (si presume di usare codici ASCII):

```
PROGRAM

VAR
  INDICE:INTEGER;
```

```
BEGIN

 FOR INDICE :=1 TO 13 DO
  BEGIN
    WRITE (CHR(INDICE*2+63))
  END

END.
```

La caratteristica espressione dopo il WRITE e' facilmente comprensibile se si tiene conto che il codice ASCII della lettera "A" e' 65.

Come alternativa una soluzione meno fantasiosa allo stesso problema e' la seguente:

```
PROGRAM

VAR
  INDICE:INTEGER;

BEGIN


 FOR INDEX:=1 TO 26 DO

  BEGIN

   IF ODD(INDICE) THEN
      BEGIN
        WRITE (CHR(INDICE+64))
      END  {IF}

 END  {FOR}

END.
```

Questo programma e' risultato inutilmente complicato per la necessita' di usare la struttura BEGIN-END con il mio compilatore: in realta' e' alquanto piu' facile di quello precedente. I commenti dopo l' istruzione END evidenziano la corrispondenza delle coppie BEGIN-END: vale la pena di abituarsi ad usare commenti di questo tipo.

Anche se avete seguito facilmente fino adesso, questo programma probabilmente vi mandera' in crisi:

```
PROGRAM STRISCE(INPUT,OUTPUT);

VAR RIGA,STELLA,STRISCIA:INTEGER;

BEGIN
  FOR RIGA := 1 TO 6 DO
```

```
  BEGIN
    FOR STELLA := 1 TO 10 DO
    BEGIN
      WRITE ('*')
    END;
    FOR STRISCIA := 1 TO 20 DO
    BEGIN
      WRITE ('=')
    END;
    WRITELN ('')
  END;
  FOR RIGA := 1 TO 5 DO
  BEGIN
    FOR STRISCIA := 1 TO 30 DO
    BEGIN
      WRITE ('=')
    END;
  WRITELN ('')
  END
```

Questo programma, che stampa in modo stilizzato la bandiera americana, e' un esempio di "loop nidificati". Questo concetto e' abbastanza autoesplicativo (e' come le bambole russe), ma e' facile confondere la nidificazione dei loop, specialmente in un programma complesso come quello precedente. Si hanno dei problemi nella nidificazione, quando si perde il conto di quale istruzione END corrisponde alle istruzioni FOR: in generale mettendo un commento a ciascuna END si risolve il problema.

Un programma piu' semplice che usa loop nidificati e' il seguente che calcola le tabelline dall' 1 al 12 (non fatelo girare su una stampante, perche' consumerebbe un mucchio di carta):

```
PROGRAM TABELLINE(INPUT,OUTPUT) ;

VAR
  A,B:INTEGER;

BEGIN

 FOR A:=1 TO 12 DO

  BEGIN

   FOR B:=1 TO 12 DO

    BEGIN
      WRITELN (A,' x ',B,' = ',A*B)
    END {per il loop B}

  END {per il loop A}

END.
```

Potreste alterare il programma precedente per stampare solo le tabelline volute: cio' e' fattibile introducendo un' istruzione READLN.

L' istruzione FOR e' indubbiamente molto utile, ma non e' sempre il modo migliore per ripetere una parte di programma per un certo numero di volte, come vedremo nel prossimo capitolo.

CAPITOLO QUARTO

ISTRUZIONI REPEAT E WHILE

Il solo modo per ripetere una parte di programma e' usare la istruzione di iterazione FOR. Tuttavia questo non e' sempre il modo migliore per fare cio'.

Se dovete ad esempio scrivere un programma per calcolare la media di una lista di numeri, potete usare l' istruzione FOR come e' stato illustrato nell'ultimo capitolo; ma che cosa succede se l' utente decide che vuole la media su 2o numeri, anziche' su 10? L'unico modo per soddisfare la richiesta sarebbe quella di modificarlo ogni volta prima di utilizzarlo, oppure di prevedere che l' utente introduca il numero dei dati da utilizzare prima di introdurre i dati stessi, riscrivendo quindi il programma in questo modo:

```
PROGRAM MEDIA(INPUT,OUTPUT);

VAR
 NUM,VAL,IND,TOT:INTEGER;

BEGIN

 WRITE ('DI QUANTI NUMERI SI VUOLE LA MEDIA ? ');

 READLN (NUM);
 TOT:=0;

 FOR IND:=1 TO NUM DO

  BEGIN
   WRITE ('INTRODURRE UN VALORE ',IND);
   READLN (VAL);
   TOT:=TOT+VAL
  END  { del loop }

 WRITELN ('');
 WRITELN ('MEDIA = ',TOT/NUM)

END.
```

Questo programma e' apprezzabile, in quanto soddisfa alla esigenza prospettata, ma e' ancora di utilita' limitata, poiche' richiede che l' utente conosca in anticipo quanti dati verranno introdotti e cio' non e' sempre pratico. Cio' che realmente serve quindi e' un programma come questo:

```
PROGRAM MEDIABIS(INPUT,OUTPUT);
```

```
VAR
 TOT,VAL,NUM:INTEGER;

BEGIN

 TOT:=0;
 NUM:=1;

 REPEAT

  WRITELN ('INTRODURRE UN VALORE ',NUM);
  READLN (VAL);
  TOT:=TOT+VAL;
  NUM:=NUM+1

 UNTIL VAL=0;

 WRITELN ('');
 WRITELN ('MEDIA = ',TOT/NUM)

END.
```

Questa nuova versione del programma continua ad accettare dati fin quando non e' introdotto un valore uguale a zero e a quel punto stampa la media dei valori introdotti prima dello zero.

L' istruzione ripetitiva REPEAT-UNTIL utilizzata nel programma esegue le istruzioni comprese tra la parola REPEAT e la parola UNTIL, finche' la condizione indicata dopo UNTIL non e' soddisfatta (cioe' diventa vera). Ci sono alcuni dettagli di sintassi:

- se nel loop ci sono piu' istruzioni, queste devono essere separate da punti e virgola e, come visto in precedenza con la struttura BEGIN-END, non e' necessario il punto e virgola prima della parola UNTIL

- la condizione che segue la parola UNTIL deve terminare con un punto e virgola, a meno che non sia seguita da un altro UNTIL o da un END

- [il mio compilatore richiede che la condizione sia tra parentesi se termina con identificatore e se e' seguita da un carattere alfanumerico, analogamente alla condizione che segue un comando IF]

L' istruzione REPEAT-UNTIL e' comoda perche' elimina la necessita' di conoscere quante volte verra' eseguita la iterazione, quando si scrive il programma o quando il calcolatore comincia ad eseguirlo.

La forma piu' generica di questa istruzione e':

```
-
-
-

REPEAT

-
<istruzioni Pascal>
-

UNTIL  espressione booleana

-
-
-
```

Questo schema indica "istruzioni" Pascal, anziche' una semplice istruzione come nei comandi IF e FOR. Cio' e' dovuto al fatto che il costrutto BEGIN-END non e' necessario in questo caso (la fine dell'iterazione e' ben definita dal comando UNTIL).

Un uso comune dell' istruzione REPEAT e' la ripetizione di un intero programma. Ad esempio, se avete un programma per calcolare le radici di un'equazione di secondo grado, e' preferibile racchiudere tutto il programma in un'istruzione di REPEAT che termina solo quando l'utente ha terminato di usare il programma. Cio' perche' l'utente possa continuare a ripetere i calcoli senza dover far ripartire ogni volta l'intero programma, cosa piuttosto seccante con certe macchine o certi compilatori.

L' istruzione REPEAT-UNTIL puo' essere simulata in Basic in questo modo:

```
-
-
100  REM  REPEAT
-
-
-
200  IF NOT(condizione) THEN GOTO 100
-
-
```

Se conoscete il Basic, potete notare che in questo programma le istruzioni da 100 a 200 sono eseguite sia che la condizione alla linea 200 sia vera, sia che sia falsa. Cio' puo' provocare problemi in alcuni programmi in cui le istruzioni non devono essere eseguite se la condizione non e' corretta. Un semplice modo per aggirare il problema e' quello di mettere il blocco REPEAT-UNTIL in un' istruzione di IF :

```
IF condizione THEN
```

```
BEGIN

  REPEAT
  istruzioni
  UNTIL NOT(condizione)

END
```

Il programma non inizia ad eseguire le istruzioni entro il blocco REPEAT-UNTIL, se la condizione e' vera. Questo sistema e' un po' contorto, per cui e' stato creato un altro tipo di istruzione iterativa:

WHILE condizione **DO** istruzione

Come al solito se si vuole piu' di un'istruzione dopo il DO deve essere usata la struttura BEGIN-END. [E al solito il mio compilatore richiede obbligatoriamente il costrutto BEGIN-END in tutti i casi].

Con il comando WHILE l'istruzione o le istruzioni che seguono il DO sono eseguite solo se la condizione e' vera, mentre se l'istruzione e' falsa il calcolatore non considera la seconda parte. Questo ci consente di scrivere istruzioni del tipo "WHILE" errore "DO" correzione. Se si usasse un REPEAT anziche' un WHILE, il programma cercherebbe di eseguire la correzione anche quando non c'e' l' errore.

I principianti di solito hanno difficolta' nel decidere quale istruzione di iterazione usare in ciascuna situazione, poiche' queste istruzioni apparentemente sono molto simili fra di loro. Il modo piu' facile per decidere quale usare e' di esprimere il problema a parole cercando di usare sia la parola MENTRE sia la parola FINCHE': una delle due espressioni suonera' errata, e questo vi dara' un' indicazione. Nell' esempio precedente i due modi di descrivere il problema sono:

- MENTRE il sistema non opera correttamente cerca di correggerlo
- cerca di correggere il sistema FINCHE' non c' e' piu' errore.

La seconda di queste descrizioni ha effetto solo se il sistema non sta lavorando quando inizia l' esecuzione del loop: quindi si capisce immediatamente che si deve usare l' istruzione WHILE.

CAPITOLO QUINTO

PROGRAMMI

CURVE SINUSOIDALI ANIMATE

Questo programma disegna sul vostro schermo una sinusoide in movimento:

```
PROGRAM  SENO(INPUT,OUTPUT) ;

CONST
   AMP = 50;

VAR
   CENTRO, FATT, ANG, SPAZI, LOOP : INTEGER;

BEGIN

  CENTRO := AMP DIV 2;
  FATT   := CENTRO-2;

  REPEAT

    FOR ANG := 0 TO 43 DO
      BEGIN
      SPAZI := TRUNC(SENO(ANG/7)*FATT)+CENTRO;

      FOR LOOP := 1 TO SPAZI DO
        BEGIN
        WRITE (' ');  {uno spazio}
        END;
      WRITELN ('+');
      END

  UNTIL  FALSE
END.
```

Per usare il programma nel vostro calcolatore dovete inserire l'ampiezza in caratteri (AMP) del vostro schermo nella sezione CONST.

Il programma opera stampando una sequenza continua di segni "+" preceduti da spazi: il numero degli spazi e' proporzionale alla curva sinusoidale.

Questo programma gira senza problemi anche sul mio compilatore. Si noti la funzione TRUNC: deve essere usata per evitare che il risultato reale del calcolo del SENO sia assegnato alla variabile SPAZI, poiche' cio' causerebbe un errore essendo SPAZI variabile intera.

CONVERSIONE DI MISURE DI TEMPERATURA

Questo programma consente di convertire una temperatura in centigradi nell'equivalente temperatura in fahrenheit.

```
PROGRAM CONV(INPUT,OUTPUT);

VAR
   FAR, CENT : REAL;

BEGIN

  WRITE  ('Centigradi ? ');
  READLN  (CENT);
  FAR  := 1.8*CENT+32;
  WRITELN  ('Fahrenheit  =  ',FAR:8:2)

END.
```

[Usando il mio compilatore si deve omettere la formattazione numerica nell'istruzione WRITELN].

EQUAZIONI DI SECONDO GRADO

Questo semplice programma risolve un' equazione del tipo "A*X*X+B*X+C=0", supponendo che tutte le radici siano reali.

```
PROGRAM  QUAD(INPUT,OUTPUT);

VAR
  A,B,C : REAL;

BEGIN

  WRITE  ('INTRODURRE a,b,c');
  READLN  (A,B,C);

  WRITELN ('Le radici sono: ',
          (-B+SQRT(B*B-4*A*C))/(2*A),
  '  e  ',(-B-SQRT(B*B-4*A*C))/(2*A) )

END.
```

Potete anche modificare il programma ponendolo dentro una istruzione di REPEAT per poter risolvere piu' equazioni in successione.

CAPITOLO SESTO

PROCEDURE DEFINITE DALL'UTENTE

In tutti i linguaggi la meccanica della programmazione fa si che si debba ragionare contemporaneamente sul programma e sull'algoritmo (l'algoritmo e' il metodo che si usa per risolvere qualche problema: per esempio un algoritmo per convertire radianti in gradi e' dato alla fine del capitolo sulle funzioni). Questo fatto puo' risultare molto pesante in programmi lunghi scritti in Basic: ad esempio se si considera il listato del mio compilatore riportato alla fine di questo libro, ci vuole molto tempo per capire come funziona, poiche' l'algoritmo e' stato reso meno evidente dalla codifica Basic.

Questo problema si risolve evitando la necessita' di codificare il programma completamente e scrivendo nel calcolatore solo l'algoritmo. Il Pascal e' l'unico linguaggio che si avvicina a questo ideale: vi consente quindi di risolvere un problema con il calcolatore allo stesso modo con cui lo risolvereste normalmente.

Facciamo un esempio. Se vi chiedessero di andare a comprare un etto di caramelle, inconsciamente suddividereste l'incarico in compiti elementari; l'incarico quindi potrebbe essere espresso nel seguene modo:

1. prendi i soldi
2. vai in pasticceria
3. scegli le caramelle
4. paga
5. torna a casa

Cio' e' logico e facile da capire.

Se poi vi chiedessero di scrivere un programma per controllare un robot che dovesse fare le stesse cose, la codifica sarebbe molto meno chiara :

1. verifica che le batterie siano cariche
2. manda un segnale alle tue gambe fornendo i dati per andare fino alla porta
3. apri la porta
4. attraversala
5. chiudi la porta
6. dai istruzioni alle gambe per camminare fino alla pasticceria

Anche questa e' una semplificazione.

Il Pascal consente di codificare lo stesso programma nel seguente modo:

```
BEGIN

 PRENDERE_SOLDI;
 ANDARE_NEGOZIO;
 SCEGLIERE_CARAMELLE;
 PAGARE_CARAMELLE;
 TORNARE_A_CASA;

END.
```

Il programma e' ora immediatamente comprensibile anche a chi non ha mai visto un calcolatore prima.

Questo programma e' scritto definendo alcune nuove procedure con nomi come "PRENDERE_SOLDI": queste nuove procedure possono quindi essere usate allo stesso modo di WRITE e READ.

A sua volta la definizione di ciascuna procedura puo' essere un elenco di altre procedure; ad esempio "TORNA_A_CASA" puo' essere formata da:

```
ESCI_DAL_NEGOZIO
PERCORRI_LA_STRADA
APRI_LA_PORTA
ENTRA_IN_CASA
CHIUDI_LA_PORTA
```

Si noti che le procedure definite dall'utente consentono di programmare e gestire indipendentemente tutte le attivita' che fanno parte di un programma. Procedendo in questo modo si ha anche il vantaggio di poter provare le procedure separatamente prima di metterle insieme per formare il programma.

LA DEFINIZIONE DI PROCEDURE

Vi sono alcuni tipi differenti di procedure definite dall' utente. Per il momento ci occuperemo solo del caso piu' semplice.

Le procedure possono ricevere degli argomenti come fa la istruzione WRITE o possono essere richiamate da sole senza argomenti, come negli esempi appena fatti. Le due alternative si differenziano per la presenza o l'assenza di una coppia di parentesi dopo il nome della procedura. Si noti che non si puo' richiamare una procedura che richiede degli argomenti senza passarle gli argomenti e viceversa.

La definizione delle procedure e' una parte della sezione dichiarativa di un programma. Le definizioni appaiono tra la

sezione VAR e il BEGIN. Ciascuna definizione di procedura deve terminare con un ";".

Il caso generale di una definizione di procedura e' :

```
PROCEDURE <identificatore>;

CONST
-
-
VAR
-
-
BEGIN
-
-
END;
```

La parola PROCEDURE avverte il compilatore della presenza della definizione di una procedura e quindi e' analoga a VAR e CONST.

L' <identificatore> e' il nome della procedura; il nome deve seguire le regole date nel capitolo 1 per la descrizione del nome di un programma.

Le sezioni CONST e VAR di una procedura possono non esservi: esse definiscono tutte le variabili e le costanti necessarie nella procedura.

La parola BEGIN indica l'inizio delle istruzioni che costituiscono la procedura. Queste istruzioni terminano con la parola END seguita da un ";".

Una procedura puo' accedere a variabili definite nel programma principale (ecco perche' la definizione delle procedure viene dopo la definizione di VAR e CONST). Potrebbe sembrare inutile che la procedura abbia le sue proprie variabili e costanti. La ragione e' che una volta che la procedura e' stata eseguita le variabili dichiarate al suo interno sono eliminate liberando la relativa memoria del calcolatore per altri scopi. Quindi l'uso di variabili "locali" e' utile per risparmiare memoria . Una ragion d'essere piu' importante e' che esse consentono la ricorsivita': questo concetto verra' spiegato in un prossimo capitolo.

Come esempio ecco una procedura per ripulire lo schermo del video (ciascun tipo di video ha un suo codice specifico per ripulire automaticamente lo schermo: il metodo usato in questa procedura funziona su qualunque terminale video).

```
PROCEDURE CLS;

VAR
  LOOP : INTEGER;

BEGIN

  FOR LOOP := 1 TO 50 DO WRITELN

END;
```

La procedura stampa 50 righe piene di spazi. Si stampano 50 linee perche' pochi terminali hanno un video piu' lungo. Un' effettiva procedura di tipo CLS (clear-screen = pulisci schermo) farebbe anche tornare il cursore all'estremita' sinistra dello schermo, cosa che non si puo' fare senza sapere il tipo di terminale usato.

Per eseguire questa procedura basta battere CLS in un programma. Quando il compilatore incontra il nome di una procedura salta automaticamente alla parte di codice che la definisce. Quando la procedura e' finita ritorna al programma iniziale da cui era venuto.

Vi sono ovviamente forti similitudini fra le procedure Pascal e le subroutines Basic. Molti programmatori in Basic usano subroutines per parti di codifica che devono essere eseguite piu' volte in un programma. Usare allo stesso modo le procedure Pascal significa sprecare un' importante possibilita'.

E' perfettamente lecito chiamare una procedura dall' interno di un' altra. In seguito vedremo anche cosa succede a chiamare una procedura dal suo stesso interno.

Alcuni compilatori PASCAL consentono di scrivere alcune procedure come routine di libreria; cio' significa che possono essere incluse in altri programmi senza riscriverle.

Per esempio un'utile procedura e' quella che consente di fare il sort di una lista di dati. Questa puo' essere dichiarata come una procedura di libreria e usata in tutti i programmi.

Un' utile proprieta' delle procedure rispetto alla semplice sequenza di codifica che esse comprendono all'interno, e' che una parte di codifica puo' essere esaminata e si puo' trovare realmente una risposta alla domanda " come funziona ? " . La domanda " che cosa fa ? " richiede un esame piu' approfondito. Se la chiamata a una procedura e' sostituita alla semplice codifica, un identificatore ben scelto consente di avere una risposta immediata alla seconda domanda. Quando si debba modificare o correggere programmi di altri autori questa e' un' utile proprieta'.

Possiamo facilmente modificare la procedura CLS per stampare un numero variabile di linee bianche:

```
PROCEDURE BLANK;

VAR
  LOOP : INTEGER;

BEGIN

  FOR LOOP := 1 TO LINES DO WRITELN

END;
```

Se questa procedura e' incorporata in un programma, alla variabile LINES deve essere associato il numero di linee bianche che devono essere stampate. Questo puo' essere un inconveniente per una routine di libreria poiche' e' necessario che il programma che la richiama non usi la variabile LINES per altri scopi. La soluzione a questo problema e' quella di consentire che LINES sia trasferita come un parametro alla procedura. Come esempio si puo' definire questa procedura:

```
PROCEDURE BLANKS (LINES : INTEGER);

VAR
  LOOP : INTEGER;

BEGIN

  FOR LOOP := 1 TO LINES DO WRITELN

END;
```

Ora la procedura puo' essere chiamata in uno di questi modi:

```
BLANKS(10)
BLANKS(HELLO)
BLANKS(3+A)
```

Quando il compilatore incontra una chiamata a una procedura con dei parametri, esso copia il valore contenuto nelle parentesi nella variabile dichiarata nell'intestazione della procedura e verifica anche il tipo di argomento della chiamata rispetto a quello specificato nell'intestazione. E' importante notare che un cambiamento fatto al valore di LINES nell'interno della procedura non si riflette nel valore della variabile usata per chiamare la procedura. Cio' e' ovvio considerando che la procedura puo' essere chiamata con un'espressione anziche' con una variabile; questa espressione puo' essere un numero, e sarebbe impossibile cambiare il valore del numero per seguire un cambiamento di LINES.

Tutto cio' e' molto bello, ma non funziona quando si vuole

scrivere una procedura per fare qualcosa come scambiare i valori di due variabili. Guardando la seguente procedura potreste pensare che possa risolvere il problema, ma, se voi la provate, vedrete che non lo fa:

```
PROCEDURE SWAP(A,B : INTEGER);

VAR
  TEMP : INTEGER;

BEGIN

  TEMP := A;
  A := B;
  B := TEMP

END;
```

La ragione per cui questa procedura fallisce e' che, malgrado tutte le variabili siano indicate correttamente, i cambiamenti non si riflettono sulle variabili usate per chiamare la routine (come sottolineato sopra questa procedura si potrebbe anche richiamare con valori "non variabili").

La procedura seguente invece esegue correttamente quanto richiesto:

```
PROCEDURE  SWAP(VAR A,B : INTEGER);

VAR
  TEMP : INTEGER;

BEGIN

  TEMP := A;
  A := B;
  B := TEMP

END;
```

La sola differenza fra questa procedura e la precedente consiste nella presenza della parola VAR nella parentesi dopo l'identificatore di procedura. Quando le variabili nell'intestazione di una procedura sono precedute dalla parola VAR esse diventano equivalenti alle variabili usate nel richiamare il programma. Cio' significa che qualunque modifica fatta alle variabili nell'interno della procedura si riflettera' nelle variabili usate nell'istruzione di chiamata alla procedura.

Lo svantaggio di questo approccio e' che non si possono usare espressioni per chiamare la procedura, ma si possono solo usare variabili del tipo corretto.

Si possono mescolare i differenti tipi di argomenti di una procedura separando le diverse sezioni con ";":

PROCEDURE TANGENTI(ARG : REAL; QUAD : INTEGER);

PROCEDURE TITOLO(A : CAR; **VAR** HE : REAL);

Quindi, diversamente dalle subroutines Basic, le procedure Pascal possono comportarsi in qualche modo come funzioni, in quanto possono prendere un argomento e lasciare un risultato; non e' pero' possibile includere chiamate a procedure in espressioni.

Il Pascal vi permette di definire sia procedure, sia funzioni : cio' e' spiegato nel prossimo capitolo.

CAPITOLO SETTIMO

FUNZIONI DEFINITE DALL'UTENTE

Le funzioni sono dichiarate praticamente nello stesso modo delle procedure:

FUNCTION <nome><parametri>:<tipo>;

La sezione <nome> dichiara il nome della funzione nel solito modo. Deve seguire le regole standard per gli identificatori.

La sezione <parametri> e' identica a quella delle procedure. Cosi' come nelle procedure, la sezione parametri puo' essere omessa se la funzione non ha argomenti.

La sezione <tipo> definisce il tipo del risultato della funzione; non puo' essere omessa.

Il resto della dichiarazione della funzione e' identico alla struttura di una definizione di procedura.

Nel definire una funzione si deve dichiarare il risultato; la definizione ha il formato:

<nome> := <tipo>

Dove <nome> e' il nome della funzione e <risultato> e' il risultato della funzione (che dovra' essere del tipo specificato nell'intestazione della definizione).

Una semplice funzione puo' essere :

```
FUNCTION SIMPLE:INTEGER;

BEGIN
  SIMPLE := 34;
END;
```

La funzione non riceve nessun argomento in ingresso, ma restituisce sempre il valore 34. Un semplice cambiamento nel programma potrebbe consentire di costruirsi la costante PI (anche se in un modo un po' lento e contorto).

La precedente lista di funzioni aveva una evidente omissione dal punto di vista dei programmatori abituati al Basic: il Pascal non ha una funzione che generi numeri a caso.

Come saprete i generatori di numeri casuali di un calcolatore danno in realta' dei numeri pseudo casuali: i numeri pseudo casuali appartengono ad una lunga sequenza che pero' si ripete

dopo un certo tempo. E' impossibile realizzare una vera casualita', ma e' sufficiente per i nostri scopi questo generatore di numeri casuali in Pascal:

```
FUNCTION RAND(VAR BASE : INTEGER) : REAL;

BEGIN

  RAND := BASE/65535;
  BASE := (25173*BASE+13849) MOD 65536

END;
```

Per usare la funzione dovete scegliere un valore iniziale per ricavare da questo una sequenza. Normalmente il programma che usa un generatore di numeri casuali chiede all'utente di introdurre l'ora o qualche altro evento casuale.

RAND deve sempre essere chiamata con questa base come argomento. Essa restituisce un numero compreso fra 0 e 1 (o che eventualmente li include). Questo numero puo' essere facilmente moltiplicato per un fattore di scala e trasformato in numero intero per simulare ad esempio il lancio di un dado. La funzione cosi' com'e' potrebbe causare overflow in un calcolatore non in grado di gestire interi a 32 bit.

Il problema di fornire una base puo' essere evitato con qualche sottigliezza di programmazione.

L'uso principale di una tale funzione e' nei giochi che richiedono eventi casuali. Questi giochi spesso richiedono che l'utente legga varie pagine di istruzioni prima che il gioco cominci. Quando l'utente termina le istruzioni normalmente deve battere il tasto RETURN. Basta quindi eseguire un loop di WHILE attendendo il RETURN e nel loop incrementare la base; poiche' il tempo necessario per leggere le istruzioni ha una grande variabilita', anche la base avra' la sua casualita'.

Le funzioni definite dall'utente sono di uso limitato per noi con la nostra conoscenza limitata del PASCAL, poiche' in molti casi si richiedono in restituzione piu' valori. Vedremo in seguito come superare questa difficolta'.

Un modo corretto di impiegare le funzioni e' il seguente:

```
REPEAT
  GIOCO
UNTIL FINE_GIOCO
```

FINE_GIOCO puo' essere quindi una funzione che restituisce un valore booleano (vero o falso) a seconda che il gioco sia o meno terminato.

Il prossimo capitolo e' dedicato a vari programmi di esempio che danno un' idea piu' chiara sull'uso delle funzioni.

CAPITOLO OTTAVO

PROGRAMMI

METODO DI NEWTON-RAPHSON

Questo programma affronta uno dei problemi principali della analisi numerica, in quanto risolve la maggior parte delle equazioni nella forma:

f(x)=0

trovando 'x' quando 'f(x)' e' uguale a zero. Scegliendo attentamente f(x) e' quindi possibile risolvere la maggior parte delle equazioni.

L' equazione f(x) e' contenuta nella definizione di funzione del programma seguente e volendo puo' essere cambiata.

Un esempio di equazione e':

f(x) = x*x-2

Si puo vedere che se f(x) e' zero , 'x' deve essere la radice quadrata di 2, cosicche' il programma, in questa forma, trova le radici quadrate. Altri numeri possono essere messi al posto di 2.

```
PROGRAM RAPHSON (INPUT, OUTPUT);

VAR X,S,START,ERROR,T,B : REAL;

FUNCTION F(X:REAL):REAL;
 BEGIN
   F := X*X-2
 END;

BEGIN

 WRITE ('dare punto iniziale : ');
 READLN (START);
 WRITE ('dare errore massimo : ');
 READLN (ERROR);

 S := START
 X := S;
 WHILE ABS (F(X)) >= ERROR DO
  BEGIN
   T := F(X) ;
   X := X+0.00001 ;
```

```
   B := (F(X)-T)/0.00001 ;
   S := S-T/B;
   X := S;
   WRITELN (X)
  END

 WRITELN  ('soluzione = ',X)

END.
```

Quando si fa girare il programma vengono richiesti due numeri. Il primo e' un punto di inizio del programma per trovare la radice. Dando sempre 1 normalmente si hanno buoni risultati. Bisogna poi introdurre l'errore massimo: e' necessario dare un numero molto piccolo come ad esempio 0.00001. Se poi si vuole una risposta piu' accurata basta diminuire l'errore di un fattore 10.

CAPITOLO NONO

TIPI DI DATI DEFINITI DALL'UTENTE

Il fatto che le variabili dichiarate siano di uno dei tipi spiegati nei precedenti capitoli dipende soltanto dalla versione del Pascal che si usa. E' anche possibile definire i propri tipi di variabili che possono essere costruiti nel modo piu' adatto ai problemi da trattare.

I nuovi tipi di variabili sono indicati in una sezione dichiarativa posta tra le sezioni CONST e VAR. (La dichiarazione di tipo puo' anche apparire nella sezione dichiarativa di una procedura o di una funzione: in questo modo e' possibile avere tipi di dati locali).

La dichiarazione di un tipo scalare e' semplicemente una lista di valori che una variabile di questo tipo puo' assumere. La dichiarazione

```
TYPE RISPOSTA = (SI,NO);
```

stabilisce che una variabile del tipo RISPOSTA puo' assumere solo i due valori SI o NO e nessun altro. Questo nuovo tipo puo' essere usato al solito modo nelle sezioni VAR:

```
VAR A:RISPOSTA;
```

E' possibile combinare le due dichiarazioni:

```
VAR  A:(SI,NO);
```

Questa seconda forma generalmente si evita poiche' non contiene la parola RISPOSTA che e' un utile puntatore per le finalita' di questo tipo.

Altri esempi di tipi scalari sono:

```
TYPE

SALUTI = (HELLO,CIAO,BONJOUR);
COLORI = (BLU,ROSSO,GIALLO);
NOTE = (DO,RE,MI,FA);
STAGIONI= (PRIMAVERA,ESTATE,AUTUNNO,INVERNO);
```

Un valore non puo' appartenere a piu' di un tipo, quindi la seguente dichiarazione e' illegale:

```
TYPE

FELINI   =  (LEONE,TIGRE,GATTO,LINCE) ;
ANIMALI  =  (CANE,GATTO,GALLINA) ;
```

I nomi dei valori listati nella dichiarazione di un tipo scalare sono costanti di quel tipo. Quindi possiamo scrivere:

```
OGGI := LUNEDI';
RELAZIONE := CUGINO;
```

Non sono permesse assegnazioni miste; cio' significa che si puo' solo assegnare il valore di tipo FELINO ad una variabile di tipo FELINO.

Non e' possibile eseguire operazioni matematiche come somma e sottrazione su variabili di tipo scalare, ma si possono eseguire dei test con operatori di relazione (< , > , etc). Quindi si ha LUNEDI' < MARTEDI'.

Le funzioni PRED e SUCC possono essere applicate a variabili di tipo scalare. Quindi PRED(MARTEDI') = LUNEDI' e SUCC(SABATO) = DOMENICA, supponendo che sia stato definito un tipo scalare dei giorni della settimana.

SUCC dell'ultimo componente della lista non e' definito ne' lo e' PRED del primo numero.

La funzione ORD da' la posizione del suo argomento nella lista; ORD(LUNEDI') e' 0, poiche' si comincia a contare da zero.

Gli scalari possono essere usati in molte applicazioni come le normali variabili, ma non possono essere usati nei comandi di READ o WRITE: tutto cio' che potreste fare e' dare il comando di WRITE del valore di ORD di una variabile scalare.

****

Un tipo "sottoinsieme" e' definito da due costanti (che possono essere a loro volta costanti scalari). Per esempio:

```
TYPE

INDICE  =  1 . . 20;
```

Con cio' si dichiara un intero (poiche' le costanti usate nella dichiarazione sono intere) la cui gamma di variazione e' limitata.

Le due costanti usate devono essere come in questo caso dello stesso tipo e di valore differente (la prima deve avere valore minore).

Si noti che sottoinsiemi di REAL non sono permessi.

Il tipo scalare associato a un sottoinsieme e' il tipo di costante usato per dichiararlo.

Variabili di sottoinsieme sono dichiarate al solito modo nelle sezioni VAR:

VAR

LOOP : INDICE;

Come per le dichiarazioni scalari, le dichiarazioni TYPE e VAR possono essere combinate in una sezione VAR (ma questo non e' sempre molto saggio).

Ogni operatore che puo' essere usato in una variabile di un particolare tipo puo' anche essere usato con un sottoinsieme di quel tipo.

Quindi una variabile di tipo INDICE si comporta esattamente come una normale variabile indice, eccetto che per le restrizioni poste sui valori che puo' assumere.

READ e WRITE (e le loro derivate) possono solo essere usate in sottoinsiemi di tipo INTEGER o CHAR.

Un uso sottile di sottoinsiemi in alcune applicazioni e' la verifica di un campo. Per esempio se vogliamo essere sicuri che un valore introdotto dall'utente in un programma e' un intero compreso in qualche specifico campo di valori, dobbiamo solo definire un tipo di sottoinsieme e quindi fare introdurre all'utente il valore direttamente nella variabile di sottoinsieme. Se il valore introdotto e' al di fuori dei limiti specificati, si avra' come risultato un errore.

L'istruzione IF e' utile per scegliere una delle due possibili alternative di azione in funzione del valore dell'espressione booleana. L'istruzione CASE e' un'istruzione di IF di tipo generalizzato; essa abilita a scegliere una fra piu' azioni in funzione del valore di un'espressione scalare di sottoinsieme.

Per esempio supponiamo che ci sia un cinema con i seguenti prezzi del biglietto:

```
RAGAZZI  : - `1000
STUDENTI : - `2000
ADULTI   : - `4000
```

L'istruzione CASE puo' essere usata per calcolare il biglietto:

TYPE

TIPO_PERSONA = (RAGAZZO,STUDENTE,ADULTO);

VAR

PREZZO : INTEGER;
PERSONA : TIPO_PERSONA;

```
BEGIN
-
-
-
CASE PERSON OF
  RAGAZZO: PREZZO := 1000;
  STUDENTE: PREZZO := 2000;
  ADULTO : PREZZO := 4000;
END;
WRITELN  ('Il prezzo e'', PREZZO)
END.
```

La variabile PERSONA nell'istruzione CASE e' chiamata selettore. I differenti valori che il selettore puo' avere sono stati usati nell'istruzione CASE come etichette. Dopo ciascuno c'e' un'istruzione. L'istruzione CASE permette ai programmi come quello sopra di essere scritti molto piu' chiaramente che con l'istruzione IF.

Se si deve usare piu' di un'istruzione dopo l'etichetta di un CASE si deve usare la costruzione BEGIN-END.

E' possibile scrivere istruzioni CASE come la seguente, che ha piu' di un' etichetta di CASE per ogni istruzione (si noti che un dato puo' apparire solo in un'etichetta):

```
CASE CODICE OF
LUNEDI': ;
MARTEDI', MERCOLEDI' : ------;
END;
```

Uno dei precedenti labels punta a un'istruzione vuota. Cio' e' permesso e puo' essere utile per eliminare la necessita' di codificare tutta l'istruzione CASE in un'altra istruzione di IF.

CAPITOLO DECIMO

MATRICI

Molti programmi richiedono la manipolazione di grandi quantita' di dati. Per esempio un tipico problema puo' essere quello di fornire al calcolatore i risultati degli esami di 345 studenti e poi dire al calcolatore di indicare un valore di soglia per cui il 60% degli studenti sia promosso.

Il problema critico e' che devono essere elaborati 345 diversi insiemi di dati. Il solo modo per mantenere questa quantita' di informazioni nel calcolatore e' di dichiarare 345 variabili di tipo intero ciascuna delle quali contiene uno dei voti.

Supponendo di poter introdurre una dichiarazione di VAR adeguata, questa soluzione puo' andar bene per la maggior parte dei casi. Il problema nasce quando i voti debbono essere introdotti nel calcolatore. Si devono usare 345 istruzioni READLN!

Il tipo array nel PASCAL e' un insieme di qualunque numero di variabili dello stesso tipo, tutte indicate con lo stesso nome.

Per esempio una matrice chiamata VOTI puo' essere usata per contenere i VOTI dell'esempio precedente. Per dire al calcolatore quali dei 345 possibili VOTI di ciascun individuo e' quello a cui vogliamo riferirci si usa un indice. L'indice e' una coppia di parentesi quadre che contiene un numero compreso fra 1 e 345.

VOTO[3] sara' il voto del terzo studente, VOTO[45] quello del 45' studente e cosi' via. Questi sono tutti elementi della matrice VOTI.

Le variabili sono dichiarate nella sezione VAR di un programma nel modo che segue:

```
VAR

VOTI : ARRAY [1..345] OF INTEGER;
```

La sezione VAR deve dire al calcolatore il nome della matrice e i limiti dei suoi indici.

Una volta che una matrice e' stata dichiarata, gli elementi individuali possono essere trattati come semplici variabili di tipo INTEGER:

```
VOTO[0] := 324-ROSSI;
VOTO[345] := VOTO[2]-1;
```

Questo non e' comunque un uso importante delle matrici. Si entra nell'uso piu' comune quando sono combimate in iterazioni con FOR, poiche' allora diventa possibile leggere in tutti i 345 voti con un programma del tipo:

```
FOR LOOP := 1 TO 345 DO

BEGIN

  WRITE ('Introdurre il voto dello studente',LOOP);
  READLN  (VOTI[LOOP])

END;
```

Una codifica analoga puo' essere usata per fare la media della matrice o per averne una lista.

Un programma completo di questo tipo dovrebbe far uso di costanti. Per esempio e' utile che, se lo stesso programma deve essere usato nella successiva sessione di esami, il numero degli studenti possa essere cambiato.

Senza un'istruzione del tipo :

```
CONST

STUDENTI = 345;
```

le necessarie modifiche sarebbero noiose.

Diversamente dal BASIC il Pascal consente di rendere un' intera matrice uguale ad un'altra mediante una linea di tipo:

```
VOTI := VECCHI_VOTI;
```

(Presupponendo che entrambe le variabili siano matrici delle stesse dimensioni).

Le matrici possono essere anche usate come parametri per procedure e funzioni.

Quindi le parentesi quadre sono necessarie solo se il programma fa riferimento a un particolare elemento della matrice.

APPENDICE 1

L'USO DEL MIO COMPILATORE

Il compilatore presentato in questa appendice e' piuttosto sofisticato e da' buoni risultati per quello che e' previsto che faccia. E'importante ricordare che non vuole sostituirsi ad un compilatore commerciale, ma e' perfetto per imparare i punti basilari della sintassi del Pascal.

Il compilatore si presenta sotto forma di due programmi in Basic uno nel diffusissimo Basic Microsoft l'altro nel popolare Basic Sinclair dello ZX Spectrum.

Le due versioni operano nello stesso modo.

Il programma Pascal va scritto all'interno di istruzioni DATA poste in testa al compilatore prima della riga 1000.

Le istruzioni RUN 1000 o GOTO 1000 consentono di far partire i miei programmi, che convertono un programma Pascal in un programma Basic (numerato a partire dalla linea 10).

Il programma Basic ottenuto dalla compilazione del programma Pascal e' semplicemente stampato: non e' stato fatto nessun tentativo di rintrodurlo nel calcolatore: cio' poiche' la procedura e' differente a seconda della macchina.

I proprietari dello ZX Spectrum e gli utenti del Basic microsoft dovranno ribattere manualmente il programma Basic nel loro calcolatore per poterlo far girare. I piu' esperti potranno farsi delle routines per introdurre automaticamente il codice tradotto.

******

La descrizione seguente e' basata sulla versione Basic Microsoft.

Il compilatore funziona su qualunque calcolatore che abbia le seguenti caratteristiche:

- nomi di variabili a due caratteri
- linee a piu' istruzioni
- matrici di stringhe
- matrici numeriche

READ
DATA
REM
GOSUB
DIM

END
RESTORE
PRINT
IF...THEN...
GOTO
FOR...=...TO...
NEXT...
RETURN
LEN
MID$
AND
OR
NOT
DEF FN....
CHR$
LEFT$
RIGHT$
INT
ABS
ATN
COS
EXP
LN
ASC
SQR

Ciascun compilatore occupa circa 24 K incluse le molte istruzioni di REM: se decidete di ometterle quando dovete introdurre il programma dovete assicurarvi che le REM puntate da istruzioni di GOTO o GOSUB siano presenti anche se vuote.

Il compilatore funziona esaminando il programma Pascal carattere per carattere e costruendo il corrispondente programma Basic.

Questo processo non e' difficile; la cosa piu' complicata e' l'analisi sintattica del programma da tradurre: quindi, anche se il compilatore non e' il modo piu' efficiente per far girare un programma in Pascal, puo' verificare se sapete usare bene il Pascal prima di comprare un vero compilatore Pascal.

Il compilatore per lo ZX Spectrum e' identico per molti aspetti alla versione Microsoft, eccetto che per le limitazioni nel trattamento di stringhe che si hanno nel Basic del Sinclair.

La versione del Pascal trattata dal compilatore comprende i seguenti comandi:

AND
BEGIN
CONST
DIV - Vedere MOD
DO

DOWNTO
ELSE
END
FOR
IF
MOD - consentito solo se la vostra versione del Basic puo' *trattare MOD.*
NOT
OR
REPEAT
THEN
TO
UNTIL
VAR

TRUE
FALSE

INTEGER
BOOLEAN
REAL
CHAR

ABS
ATN
CHR
COS
EXP
LN
ODD
ORD
PRED
ROUND
SIN
SQR
SQRT
SUCC
TRUNC

APPENDICE 2

COMPILATORE PASCAL PER BASIC MICROSOFT

```
1    ************PROGRAMMA DI PROVA ***************
10 DATA "CONST P=3.14;"
20 DATA "VAR A,B,C:REAL; I:INTEGER; K:CHAR;
   SW:BOOLEAN;"
25 DATA ""
30 DATA "BEGIN"
32 DATA ""
35 DATA "  READLN(B);"
40 DATA "  FOR I:=15 DOWNTO 10 DO"
42 DATA "    BEGIN"
44 DATA "    C:=(B*P)"
46 DATA "    WRITELN('circ=',C)
50 DATA "    END;"
60 DATA ""
70 DATA "  WHILE K<>'   DO"
72 DATA "    BEGIN"
74 DATA "    READLN(K)"
76 DATA "    END;"
80 DATA ""
82 DATA "  IF (B<C)OR(C=0) THEN
84 DATA "    BEGIN"
86 DATA "    WRITE('//////////////')"
88 DATA "    END"
90 DATA "  ELSE"
92 DATA "    BEGIN"
94 DATA "    WRITE('%%%%%%%%%%%%')"
96 DATA "    END;"
98 DATA ""
100 DATA " A:=ABS(B)+ATN(B)-COS(B)*EXP(B)/
    (-3.14E-2*B);"
110 DATA " A:=ROUND(B)+SIN(B)-SQR(B)*SQRT(B)-LN(B);"
120 DATA " A:=TRUNC(B)+ODD(I)-ORD(B)*PRED(B);"
130 DATA " K:=CHR(SUCC(22));A:=B MOD C +B DIV C;"
140 DATA " SW:=NOT FALSE;"
150 DATA " SW:=NOT(1 OR SW AND TRUE)OR(NOT FALSE);"
180 DATA " WRITELN('---------- FINE TEST ----------')
190 DATA ""
```

```
200 DATA "END."
1000 '--------------------------------------------------------
1010 '
1020 '*******  COMPILATORE  PASCAL  *******
1030 '
1040 '     (c) Jeremy Ruston '82
1050
1060 '---------------------------------------------------------
1070 '
1100 '<<<<<nucleo programma<<<<<
1120 '
1130 '<<<<<input programma Pascal
1140 GOSUB 1310
1150 '<<<<<inizializza variabili
1160 DIM BA$(LE*2+8),VA$(20),TV(20),CO$(20),
     SS$(50)
1170 SP=50
1180 LP=1
1190 CP=1
1200 PV=1
1210 PC=1
1220 JA$=""
1230 JB$=""
1240 KA$=""
1250 MA$=""
1260 '<<<<<compila
1270 GOSUB 5170
1280 '<<<<<lista Basic compilato
1290 GOSUB 1500 : END
1300 <<<<<ROUTINES<<<<<
1310 '
1320 '<<<<<legge/lista programma Pascal
1330 '
1340 RESTORE
1350 PRINT "<<<<< PASCAL >>>>>":PRINT
1360 LE=1
1370 READ A$
1380 PRINT A$
1390 IF A$="END." THEN GOTO 1420
1400 LE=LE+1
1410 GOTO 1370
```

```
1420 LE=LE+1
1430 DIM PA$(LE)
1440 RESTORE
1450 FOR T=1 TO LE-1
1460 READ PA$(T):LET PA$(T)=PA$(T)+" "
1470 NEXT T
1480 PA$(LE)="**********************"
1490 RETURN
1500 '
1510 '<<<<<lista Basic
1520 '
1530 PRINT:PRINT "<<<<< BASIC >>>>>":PRINT
1540 FOR T=1 TO LB-1
1550 PRINT T*10;" ";BA$(T)
1560 NEXT T
1570 RETURN
1580 '
1590 '<<<<<incrementa puntatori Pascal
1600 '
1610 CP=CP+1
1620 IF CP>LEN(PA$(LP)) THEN CP=1:LP=LP+1
1630 RETURN
1640 '
1650 '<<<<<decrementa puntatori Pascal
1660 '
1670 CP=CP-1
1680 IF CP=0 THEN LP=LP-1:CP=LEN(PA$(LP))
1690 RETURN
1700 '
1710 '<<<<<legge 1 carattere Pascal
1720 '<<<<<incrementa puntatori
1730 '
1740 A$=MID$(PA$(LP),CP,1)
1750 GOSUB 1580
1760 RETURN
1770 '
1780 '<<<<<legge 1 carattere Pascal
1790 '<<<<<salta spazi
1800 '<<<<<incrementa puntatori
1810 '
1820 GOSUB 1700
```

```
1830 IF A$=" " THEN GOTO 1820
1840 RETURN
1850 '
1860 '<<<<< stampa messaggi errore
1870 '
1880 PRINT:PRINT:PRINT
1890 IF LP<>1 THEN PRINT PA$(LP-1)
1900 PRINT PA$(LP)
1910 IF LP<>LE THEN PRINT PA$(LP+1)
1920 PRINT ">>>>> ERRORE : ";A$;CHR$(7)
1930 END
1940 '
1950 '<<<<<salva puntatori Pascal
1960 '
1970 LG=LP
1980 CG=CP
1990 RETURN
2000 '
2010 '<<<<<ripristina puntatori Pascal
2020 '
2030 LP=LG
2040 CP=CG
2050 RETURN
2060 '
2070 '<<<<<legge 1 cifra
2080
2090 GOSUB 1940
2100 GOSUB 1770
2110 IF A$>="0" AND A$<="9" THEN RETURN
2120 GOSUB 2000
2130 A$=""
2140 RETURN
2150 '
2160 '<<<<<legge 1 carattere alfabetico
2170 '
2180 GOSUB 1940
2190 GOSUB 1770
2200 IF A$>="A" AND A$<="Z" THEN RETURN
2210 GOSUB 2000
2220 A$=""
2230 RETURN
```

```
2240
2250 '<<<<<legge 1 intero senza segno
2260 '
2270 AA$=""
2275 GOSUB 2060:GOTO 2290
2280 GOSUB 1940:GOSUB 1700:GOSUB 2110
2290 IF A$="" THEN A$=AA$:RETURN
2300 AA$=AA$+A$
2310 GOTO 2280
2320 '
2330 '<<<<<legge identificatore
2340 '
2350 BA$=""
2360 GOSUB 2150
2370 IF A$="" THEN RETURN
2380 BA$=BA$+A$
2390 GOSUB 1940:GOSUB 1700:GOSUB 2110
2400 BB$=A$
2410 BA$=BA$+A$
2420 GOSUB 1940:GOSUB 1700:GOSUB 2200
2430 IF A$="" AND BB$="" THEN A$=BA$:RETURN
2440 GOTO 2380
2450 '
2460 '<<<<<legge 1 numero senza segno
2470 '
2480 CB$=""
2490 GOSUB 2240
2500 IF A$="" THEN RETURN
2510 CB$=CB$+A$
2520 GOSUB 1940
2530 GOSUB 1700
2540 GOSUB 2000
2550 IF A$<>"." THEN GOTO 2620
2560 GOSUB 1770
2570 CB$=CB$+A$
2580 GOSUB 2280
2590 IF A$="" THEN RETURN
2600 CB$=CB$+A$
2610 GOSUB 1940
2620 GOSUB 1700
2630 IF A$<>"E" THEN A$=CB$:GOSUB 2000:RETURN
```

```
2640 CB$=CB$+A$
2650 GOSUB 1940
2660 GOSUB 1700
2670 GOSUB 2000
2680 IF A$="+" OR A$="-"THEN CB$=CB$+A$:GOSUB1700
2690 GOSUB 2240
2700 IF A$="" THEN RETURN
2710 A$=CB$+A$
2720 RETURN
2730 '
2740 'legge stringa
2750 '
2760 GOSUB 1940
2770 GOSUB 1770
2780 GOSUB 2000
2790 IF A$<>"'" THEN A$="":RETURN
2800 DB$=CHR$(34)
2810 GOSUB 1770
2820 GOSUB 1700
2830 DB$=DB$+A$
2840 IF A$<>"'" THEN GOTO 2820
2850 A$=LEFT$(DB$,LEN(DB$)-1)+CHR$(34)
2860 RETURN
2870 '
2880 '<<<<<legge costante senza segno
2890 '
2900 CD=CP:LD=LP
2910 GOSUB 2320
2920 IF A$="" THEN CP=CD:LP=LD:GOTO 3000
2930 FL=0
2940 FOR ET=1 TO PC-1
2950 IF CO$(ET)=A$ THEN FL=ET
2960 NEXT ET
2970 IF FL =0 THEN A$="":CP=CD:LP=LD:RETURN
2980 IF TC(FL)=4 THEN A$=A$+"$"
2990 RETURN
3000 GOSUB 2450
3010 IF A$<>"" THEN RETURN
3020 GOSUB 2730
3030 IF A$<>"" THEN RETURN
3040 RETURN
```

```
3050
3060 '<<<<<legge costante
3070 '
3080 GOSUB 2320
3090 IF A$="" THEN GOTO 3170
3100 FL=0
3110 FOR FT=1 TO PC-1
3120 IF CO$(FT)=A$"" THEN FL=FT
3130 NEXT FT
3140 IF FL=0 THEN A$="":RETURN
3150 IF TC(FL)=4 THEN A$=A$+"$"
3160 RETURN
3170 B$=""
3180 GOSUB 1940
3190 GOSUB 1770
3200 GOSUB 2000
3210 IF A$="+" OR A$="-" THEN B$=A$:GOSUB 1770
3220 GOSUB 2450
3230 IF A$<>"" THEN A$=B$+A$:RETURN
3240 GOSUB 2730
3250 IF A$<>"" THEN A$=B$+A$:RETURN
3260 A$="identificatore di costante non valido"
3270 GOTO 1850
3280 '
3290 '<<<<<legge variabile
3300 '
3310 LD=LP:CD=CP
3320 GOSUB 2320
3330 IF A$="" THEN LP=LD:CP=CD:RETURN
3340 FL=0
3350 FOR GT=1 TO PV-1
3360 IF VA$(GT)=A$ THEN FL=GT
3370 NEXT GT
3380 IF TV(FL)=4 THEN A$=A$+"$"
3390 IF FL<>0 THEN RETURN
3400 A$=""
3410 LP=LD:CP=CD
3420 RETURN
3430 '
3440 '<<<<<push
3450 '
```

```
3460 SS$(SP)=JA$
3470 SS$(SP-1)=JB$
3480 SS$(SP-2)=KA$
3490 SS$(SP-3)=MA$
3500 SP=SP-4
3510 RETURN
3520 '
3530 '<<<<<push A$
3540 '
3550 SS$(SP)=A$
3560 SP=SP-1
3570 RETURN
3580 '
3590 '<<<<<pull
3600 '
3610 MA$=SS$(SP+1)
3620 KA$=SS$(SP+2)
3630 JB$=SS$(SP+3)
3640 JA$=SS$(SP+4)
3650 SP=SP+4
3660 RETURN
3670 '
3680 '<<<<<pull A$
3690 '
3700 SP=SP+1
3710 A$=SS$(SP)
3720 RETURN
3730 '
3740 '<<<<<legge 1 fattore espressione
3750 '
3760 GOSUB 1940
3770 GB$=""
3780 FOR GT=1 TO 3
3790 GOSUB 1770
3800 GB$=GB$+A$
3810 NEXT GT
3830 IF GB$<>"NOT" THEN GB$="":GOSUB 2000
3840 GOSUB 2870
3850 IF A$<>"" THEN A$=GB$+" "+A$:RETURN
3860 GOSUB 3280
3870 IF A$<>"" THEN A$=GB$+" "+A$:RETURN
```

```
3880 GOSUB 1940
3890 GOSUB 1770
3900 GOSUB 2000
3910 IF A$<> "(" THEN GOTO 4040
3920 GOSUB 1770
3930 A$=GB$+"("
3940 GOSUB 3520
3950 GOSUB 3430
3960 GOSUB 4970
3970 GOSUB 3580
3980 GB$=A$
3990 GOSUB 1770
4000 IF A$<>")" THEN A$="manca parentesi chiusa":
     GOTO 1850
4010 GOSUB 3670
4020 A$=A$+GB$+")"
4030 RETURN
4040 GOSUB 1940
4050 GC$=GB$
4060 GB$=""
4070 FOR GT=1 TO 5
4080 GOSUB 1770
4090 GB$=GB$+A$
4100 NEXT GT
4110 GOSUB 2000
4120 GN=0
4130 IF LEFT$(GB$,3)="ABS" THEN GN=3:A$="ABS"
4140 IF LEFT$(GB$,3)="ATN" THEN GN=3:A$="ATN"
4150 IF LEFT$(GB$,3)="CHR" THEN GN=3:A$="CHR$"
4160 IF LEFT$(GB$,3)="COS" THEN GN=3:A$="COS"
4170 IF LEFT$(GB$,3)="EXP" THEN GN=3:A$="EXP"
4180 IF LEFT$(GB$,2)="LN"  THEN GN=2:A$="LN"
4190 IF LEFT$(GB$,3)="ODD" THEN GN=3:A$="FNOD"
4200 IF LEFT$(GB$,3)="ORD" THEN GN=3:A$="ASC"
4210 IF LEFT$(GB$,4)="PRED" THEN GN=4:A$="FNPR"
4220 IF LEFT$(GB$,5)="ROUND" THEN GN=5:A$="FNRO"
4230 IF LEFT$(GB$,3)="SIN"   THEN GN=3:A$="SIN"
4240 IF LEFT$(GB$,4)="SQRT"
      THEN GN=4:A$="SQR":GOTO 4280
4250 IF LEFT$(GB$,3)="SQR"  THEN GN=3:A$="FNSQ"
4260 IF LEFT$(GB$,4)="SUCC"  THEN GN=4:A$="FNSU"
```

```
4270 IF LEFT$(GB$,5)="TRUNC"  THEN GN=5:A$="INT"
4280 IF GN=0  THEN A$="funzione errata":GOTO 1850
4290 GC$=GC$+A$
4300 FOR GT=1 TO GN
4310 GOSUB 1770
4320 NEXT GT
4330 GOSUB 1770
4340 IF A$<>"(" THEN A$=
     "chiamata funzione senza parentesi aperta":
     GOTO 1850
4350 GC$=GC$+A$
4360 A$=GC$
4370 GOSUB 3520
4380 GOSUB 3430
4390 GOSUB 4970
4400 GOSUB 3580
4410 GC$=A$
4420 GOSUB 3670
4430 GC$=A$+GC$
4440 GOSUB 1770
4450 IF A$<>")" THEN A$=
     "funzione senza parentesi chiusa":GOTO 1850
4460 A$=GC$+")"
4470 RETURN
4480 '
4490 '<<<<<legge 1 termine espressione
4500 '
4510 GOSUB 3730
4520 IF A$=""THEN A$="omissione fattore":GOTO 1850
4530 JA$=A$
4540 JB$=""
4550 GOSUB 1940
4560 FOR JT=1 TO 3
4570 GOSUB 1770
4580 JB$=JB$+A$
4590 NEXT JT
4600 GOSUB 2000
4610 IF JB$<>"DIV" AND JB$<>"MOD" AND JB$<>"AND"
     AND LEFT$(JB$,1)<>"*" AND LEFT$(JB$,1)<>"/"
     THEN A$=JA$:RETURN
4620 JA=1
```

```
4630 IF JB$="AND"OR JB$="DIV"OR JB$="MOD"THENJA=3
4640 FOR T=1 TO JA
4650 GOSUB 1770
4660 NEXT T
4670 GOSUB 3730
4680 IF A$="" THEN A$="manca fattore":GOTO 1850
4690 JA$=JA$+" "+LEFT$(JB$,JA)+" "+A$+" "
4700 GOTO 4540
4710 '
4720 '<<<<<espressione semplice
4730 '
4740 GOSUB 1940
4750 GOSUB 1770
4760 GOSUB 2000
4770 KA$=""
4780 IF A$="+" OR A$="-" THEN KA$=A$:GOSUB 1770
4790 GOSUB 4480
4800 IF A$="" THEN A$="termine mancante":GOTO 1850
4810 KA$=KA$+A$
4820 GOSUB 1940
4830 KB$=""
4840 GOSUB 1770
4850 KB$=KB$+A$
4860 GOSUB 1770
4870 KB$=KB$+A$
4880 GOSUB 2000
4890 IF LEFT$(KB$,1)<>"+" AND LEFT$(KB$,1)<>"-"
     AND KB$<>"OR" THEN A$=KA$:RETURN
4900 IF KB$<>"OR" THEN KA$=KA$+LEFT$(KB$,1)
4910 GOSUB 1770
4920 IF KB$="OR" THEN KA$=KA$+" OR ":GOSUB 1770
4930 GOSUB 4480
4940 KA$=KA$+A$
4950 IF A$="" THEN A$=KA$:RETURN
4960 GOTO 4820
4970 '
4980 'legge espressione
4990 '
5000 GOSUB 4710
5010 MA$=A$
5020 GOSUB 1940
```

```
5030 MB$=""
5040 FOR MT=1 TO 2
5050 GOSUB 1770
5060 MB$=MB$+A$
5070 NEXT MT
5080 GOSUB 2000
5090 MC$=LEFT$(MB$,1)
5100 IF MC$<>"<" AND MC$<>"=" AND MC$<>">"
     THEN A$=MA$:RETURN
5110 IF MB$="<>" OR MB$="<=" OR MB$=">="
     THEN GOSUB 1770:GOSUB 1770:A$=MB$:GOTO 5130
5120 GOSUB 1770:A$=MC$
5130 MA$=MA$+A$
5140 GOSUB 4710
5150 A$=MA$+A$
5160 RETURN
5170 '----------------------------------------------
5180 '
5190 '
5200 '
5210 '
5220 '
5230 '<<<<routines principali di compilazione>>>>
5240
5250 '
5260 '
5270 '
5280 '----------------------------------------------
5290 '
5300 BA$(1)="DEF FNOD(X)=((X-INT(X/2)*2)=1)"
5310 BA$(2)="DEF FNPR(X)=X-1"
5320 BA$(3)="DEF FNSU(X)=X+1"
5330 BA$(4)="DEF FNRO(X)=INT(X+0.5)"
5340 BA$(5)="DEF FNSQ(X)=X*X"
5350 LB=6
5360 GOSUB 5400
5370 GOSUB 5790
5380 GOSUB 6230
5390 RETURN
5400 '
5410 '<<<<<sezione costanti<<<<<
```

```
5420 '
5430 CO$(1)="TRUE":CO$(2)="FALSE":PC=3
5440 BA$(LB)="REM ----CONST----"
5450 BA$(LB+1)="TRUE=-1"
5460 BA$(LB+2)="FALSE=0"
5470 LB=LB+3
5480 MD$=""
5490 GOSUB 1940
5500 FOR MT=1 TO 5
5510 GOSUB 1770
5520 MD$=MD$+A$
5530 NEXT MT
5540 IF MD$<>"CONST" THEN GOSUB 2000:RETURN
5550 GOSUB 2320
5560 IF A$="" THEN A$=
     "manca identificatore di costante":GOTO 1850
5570 CO$(PC)=A$
5580 GOSUB 1770
5590 IF A$<>"=" THEN A$="manca segno uguale":
     GOTO 1850
5600 GOSUB 3050
5610 IF A$="" THEN A$="manca costante":GOTO 1850
5620 IF LEFT$(A$,1)=CHR$(34) THEN TC(PC)=4
5630 BA$(LB)=CO$(PC)
5640 IF TC(PC)=4 THEN BA$(LB)=BA$(LB)+"$"
5650 BA$(LB)=BA$(LB)+"="+A$
5660 LB=LB+1
5670 PC=PC+1
5680 GOSUB 1770
5690 IF A$<>";" THEN A$=
     "manca punto e virgola":GOTO 1850
5700 GOSUB 1940
5710 MD$=""
5720 FOR MT=1 TO 5
5730 GOSUB 1770
5740 MD$=MD$+A$
5750 NEXT MT
5760 GOSUB 2000
5770 IF MD$="BEGIN" OR LEFT$(MD$,3)="VAR"
     THEN RETURN
5780 GOTO 5550
```

```
5790
5800 '<<<<<sezione variabili<<<<<
5810 '
5820 GOSUB 1940
5830 ME$=""
5840 FOR T=1 TO 3
5850 GOSUB 1770
5860 ME$=ME$+A$
5870 NEXT T
5880 GOSUB 2000
5890 IF ME$<>"VAR" THEN RETURN
5900 GOSUB 1770:GOSUB 1770:GOSUB 1770
5910 A$="END"
5920 GOSUB 3520
5930 GOSUB 2320
5940 IF A$="" THEN A$="manca identificatore":
     GOTO 1850
5950 GOSUB 3520
5960 GOSUB 1770
5970 IF A$=":" THEN GOTO 6000
5980 IF A$<>"," THEN A$=
     "manca virgolain sezione variabili":GOTO 1850
5990 GOTO 5930
6000 GOSUB 2320
6010 IF A$<>"REAL"AND A$<>"INTEGER" AND A$<>"CHAR"
     AND A$<>"BOOLEAN" THEN A$="tipo errato":
     GOTO 1850
6020 IF A$="REAL" THEN MM=1
6030 IF A$="INTEGER" THEN MM=2
6040 IF A$="BOOLEAN" THEN MM=3
6050 IF A$="CHAR" THEN MM=4
6060 GOSUB 1770
6070 IF A$<>";" THEN A$=
     "manca punto e virgola":GOTO 1850
6080 GOSUB 3670
6090 IF A$="END" THEN GOTO 6140
6100 VA$(PV)=A$
6110 TV(PV)=MM
6120 PV=PV+1
6130 GOTO 6080
6140 ME$=""
```

```
6150 GOSUB 1940
6160 FOR T=1 TO 5
6170 GOSUB 1770
6180 ME$=ME$+A$
6190 NEXT T
6200 GOSUB 2000
6210 IF ME$="BEGIN" THEN RETURN
6220 GOTO 5910
6230 '
6240 '<<<<<blocco istruzioni<<<<<
6250 '<<<<<passo 1 compilazione
6260 BA$(LB)="REM ---BLOCK---"
6270 LB=LB+1
6280 NA$=""
6290 EN=0
6300 FOR T=1 TO 5
6310 GOSUB 1770
6320 NA$=NA$+A$
6330 NEXT T
6340 IF NA$<>"BEGIN" THEN A$="manca BEGIN":
     GOTO 1850
6350 GOSUB 7090
6360 IF EN=0 THEN GOTO 6350
6370 '
6380 '<<<<<passo 2 compilazione
6390 '
6400 FOR T=1 TO LB-1
6410 A$=BA$(T)
6420 IF A$="REM ---END---" THEN GOSUB 6460
6430 IF A$="REM ---UNTIL---" THEN GOSUB 6970
6440 NEXT T
6450 RETURN
6460 LI=T-1
6470 BE=0
6480 IF BA$(LI)="REM ----BEGIN---" AND BE=0
     THEN GOTO 6540
6490 IF BA$(LI)="REM ----BEGIN---" AND BE=0
     THEN BE=BE+1
6500 IF BA$(LI)="REM ----END----" THEN BE=BE-1
6510 LI=LI-1
6520 IF LI=0 THEN PRINT
```

```
     "BEGIN-END non corrispondono":END
6530 GOTO 6480
6540 IF BA$(LI-2)="REM ---IF---" THEN GOTO 6750
6550 IF BA$(LI-2)="REM ----WHILE---" THEN GOTO 6700
6560 IF BA$(LI-2)="REM ----FOR---" THEN GOTO 6600
6570 BA$(LI)="REM BEGIN"
6580 BA$(T)="REM END"
6590 RETURN
6600 BA$(LI)="REM BEGIN"
6610 BA$(LI-2)="REM FOR"
6620 H$=""
6630 H=5
6640 H$=H$+MID$(BA$(LI-1),H,1)
6650 H=H+1
6660 IF MID$(BA$(LI-1),H,1)="=" THEN GOTO 6680
6670 GOTO 6640
6680 BA$(T)="NEXT "+U$+":REM END"
6690 RETURN
6700 BA$(LI)="REM BEGIN"
6710 BA$(LI-2)="REM WHILE"
6720 BA$(T)="GOTO "+STR$((LI-2)*10)+":REM END"
6730 BA$(LI-1)=BA$(LI-1)+STR$((T+1)*10)
6740 RETURN
6750 EL=0
6760 IF BA$(T+1)="REM ---ELSE---" THEN EL=1
6770 BA$(LI-1)="REM THEN"
6780 BA$(LI-3)=BA$(LI-3)+STR$((T+1)*10)
6790 BA$(LI-2)="REM IF"
6800 BA$(LI-4)=BA$(LI-4)+STR$((LI-1)*10)
6810 BA$(LI)="REM BEGIN"
6820 BA$(T)="REM END"
6830 IF EL=0 THEN RETURN
6840 BA$(T+1)="REM ELSE"
6850 BA$(T+2)="REM BEGIN"
6860 EN=0
6870 LL=T+3
6880 IF BA$(LL)="REM ---END---" AND EN=0
     THEN GOTO 6940
6890 IF BA$(LL)="REM ---END---" THEN EN=EN+1
6900 IF BA$(LL)="REM ---BEGIN---" THEN EN=EN-1
6910 LL=LL+1
```

```
6920 IF LL>=LB THEN PRINT"sezione ELSE errata":END
6930 GOTO 6880
6940 BA$(T)="GOTO "+STR$(LL*10)+":REM END"
6950 BA$(LL)="REM END"
6960 RETURN
6970 LI=T-1
6980 RE=0
6990 IF BA$(LI)="REM ---REPEAT---" AND RE=0
     THEN GOTO 7050
7000 IF BA$(LI)="REM ---REPEAT---" THEN RE=RE+1
7010 IF BA$(LI)="REM ---UNTIL---" THEN RE=RE-1
7020 LI=LI-1
7030 IF LI=0 THEN PRINT
     "REPEAT-UNTIL non corrispondono" :END
7040 GOTO 6990
7050 BA$(T+1)=BA$(T+1)+STR$(LI*10)
7060 BA$(LI)="REM REPEAT"
7070 BA$(T)="REM UNTIL"
7080 RETURN
7083 '
7084 '<<<<<fine compilazione<<<<<
7085 '--------------------------------------------------
7090 '
7100 '<<<<<legge istruzione
7110 '
7120 NA$=""
7130 GOSUB 1940
7140 FOR T=1 TO 6
7150 GOSUB 1770
7160 NA$=NA$+A$
7170 NEXT T
7180 GOSUB 2000
7190 IF LEFT$(NA$,5)="WRITE"   THEN GOTO 7480
7200 IF       NA$   ="READLN"  THEN GOTO 7780
7210 IF LEFT$(NA$,5)="BEGIN"   THEN GOTO 8000
7220 IF LEFT$(NA$,3)="END"     THEN GOTO 8090
7230 IF LEFT$(NA$,2)="IF"      THEN GOTO 8260
7240 IF LEFT$(NA$,4)="THEN"    THEN GOTO 8460
7250 IF LEFT$(NA$,4)="ELSE"    THEN GOTO 8550
7260 IF       NA$   ="REPEAT"  THEN GOTO 8640
7270 IF LEFT$(NA$,5)="UNTIL"   THEN GOTO 8730
```

```
7280 IF LEFT$(NA$,5)="WHILE"    THEN GOTO 8860
7290 IF LEFT$(NA$,3)="FOR"      THEN GOTO 9040
7300 '
7310 '<<<<< istruzioni di assegnazione
7320 '
7330 GOSUB 3280
7340 IF A$="" THEN A$="variabile non valida":
     GOTO 1850
7350 BA$(LB)=A$+"="
7360 GOSUB 1770
7370 IF A$<>":" THEN A$="mancano due punti":
     GOTO 1850
7380 GOSUB 1770
7390 IF A$<>"=" THEN A$="manca uguale":GOTO 1850
7400 GOSUB 4970
7410 IF A$=""THEN A$="manca espressione":GOTO1850
7420 BA$(LB)=BA$(LB)+A$
7430 GOSUB 1940
7440 GOSUB 1770
7450 IF A$<>";" THEN GOSUB 2000
7460 LB=LB+1
7470 RETURN
7480 '
7490 '<<<<<WRITE-WRITELN
7500 '
7510 FOR T=1 TO 5
7520 GOSUB 1770
7530 NEXT T
7540 GOSUB 1940
7550 GOSUB 1770
7560 NA$=A$
7570 GOSUB 1770
7580 NA$=NA$+A$
7590 IF NA$<>"LN" THEN NA$="":GOSUB 2000
7600 BA$(LB)="PRINT "
7610 GOSUB 1770
7620 IF A$<>"(" THEN A$=
      "manca parentesi aperta":GOTO 1850
7630 GOSUB 1940
7640 GOSUB 1770
7650 GOSUB 2000
```

```
7660 IF A$=")" THEN GOSUB 1770:GOTO 7720
7670 GOSUB 4970
7680 BA$(LB)=BA$(LB)+A$
7690 GOSUB 1770
7700 IF A$="," THEN BA$(LB)=BA$(LB)+A$:GOTO 7630
7710 IF A$<>")" THEN A$="carattere non valido":
     GOTO 1850
7720 IF NA$<>"LN" THEN BA$(LB)=BA$(LB)+";"
7730 LB=LB+1
7740 GOSUB 1940
7750 GOSUB 1770
7760 IF A$<>";" THEN GOSUB 2000
7770 RETURN
7780 '
7790 '<<<<<READLN
7800 '
7810 FOR T=1 TO 6
7820 GOSUB 1770
7830 NEXT T
7840 GOSUB 1770
7850 IF A$<>"(", THEN A$=
     "manca parentesi aperta":GOTO 1850
7860 BA$(LB)="INPUT "
7870 GOSUB 3280
7880 IF A$="" THEN A$="manca variabile":GOTO 1850
7890 BA$(LB)=BA$(LB)+A$
7900 GOSUB 1770
7910 IF A$=")" THEN GOTO 7950
7920 IF A$<>"," THEN A$="manca virgola":GOTO 1850
7930 BA$(LB)=BA$(LB)+","
7940 GOTO 7870
7950 GOSUB 1940
7960 GOSUB 1770
7970 IF A$<>";" THEN GOSUB 2000
7980 LB=LB+1
7990 RETURN
8000 '
8010 '<<<<<BEGIN
8020 '
8030 BA$(LB)="REM ---BEGIN---"
8040 LB=LB+1
```

```
8050 FOR T=1 TO 5
8060 GOSUB 1770
8070 NEXT T
8080 RETURN
8090 '
8100 '<<<<<END
8110 '
8120 FOR T=1 TO 3
8130 GOSUB 1770
8140 NEXT T
8150 GOSUB 1940
8160 GOSUB 1770:IF A$=";" THEN GOTO 8230
8170 GOSUB 2000
8180 IF A$<>"." THEN GOTO 8230
8190 EN=1
8200 BA$(LB)="END"
8210 LB=LB+1
8220 RETURN
8230 BA$(LB)="REM ---END---"
8240 LB=LB+1
8250 RETURN
8260 '
8270 '<<<<<IF
8280 '
8290 GOSUB 1770:GOSUB 1770
8300 GOSUB 4970
8310 IF A$="" THEN A$=
     "manca condizione dopo IF":GOTO 1850
8320 BA$(LB)="IF "+A$+" THEN "
8330 LB=LB+1
8340 BA$(LB+1)="REM ---IF---"
8350 BA$(LB)="IF NOT("+A$+") THEN "
8360 LB=LB+2
8370 ME$=""
8380 GOSUB 1940
8390 FOR T=1 TO 4
8400 GOSUB 1770
8410 ME$=ME$+A$
8420 NEXT T
8430 IF ME$<>"THEN" THEN A$=
     "manca THEN dopo IF":GOTO 1850
```

```
8440 GOSUB 2000
8450 RETURN
8460 '
8470 '<<<<<THEN
8480 '
8490 FOR T=1 TO 4
8500 GOSUB 1770
8510 NEXT T
8520 BA$(LB)="REM ---THEN---"
8530 LB=LB+1
8540 RETURN
8550 '
8560 '<<<<<ELSE
8570 '
8580 FOR T=1 TO 4
8590 GOSUB 1770
8600 NEXT T
8610 BA$(LB)="REM ---ELSE---"
8620 LB=LB+1
8630 RETURN
8640 '
8650 '<<<<<REPEAT
8660 '
8670 FOR T=1 TO 6
8680 GOSUB 1770
8690 NEXT T
8700 BA$(LB)="REM ---REPEAT---"
8710 LB=LB+1
8720 RETURN
8730 '
8740 '<<<<<UNTIL
8750 '
8760 BA$(LB)="REM ---UNTIL---"
8770 LB=LB+1
8780 FOR T=1 TO 5
8790 GOSUB 1770
8800 NEXT T
8810 GOSUB 4970
8820 IF A$="" THEN A$=
     "manca espressione dopo UNTIL":GOTO 1850
8830 BA$(LB)="IF NOT("+A$+") THEN GOTO "
```

```
8835 GOTO 7430
8860 '
8870 '<<<<<WHILE
8880 '
8890 BA$(LB)="REM ---WHILE---"
8900 LB=LB+1
8910 FOR T=1 TO 5
8920 GOSUB 1770
8930 NEXT T
8940 GOSUB 4970
8950 IF A$="" THEN A$=
     "manca espressione dopo WHILE":GOTO 1850
8960 BA$(LB)="IF NOT ("+A$+") THEN GOTO "
8970 GOSUB 1770
8980 MM$=A$
8990 GOSUB 1770
9000 MM$=MM$+A$
9010 IF MM$<>"DO" THEN A$=
     "manca DO dopo WHILE":GOTO 1850
9020 LB=LB+1
9030 RETURN
9040 '
9050 '<<<<<FOR
9060 '
9070 BA$(LB)="REM ---FOR----"
9080 LB=LB+1
9090 FOR T=1 TO 3
9100 GOSUB 1770
9110 NEXT T
9120 GOSUB 3280
9130 IF A$="" OR TV(FL)=4 THEN A$=
     "FOR con variabile non valida":GOTO 1850
9140 BA$(LB)="FOR "+A$+"="
9150 GOSUB 1770
9160 IF A$<>":" THEN A$=
     "mancano due punti dopo FOR"GOTO 1850
9170 GOSUB 1770
9180 IF A$<>"=" THEN A$=
     "manca uguale dopo FOR":GOTO 1850
9190 GOSUB 4970
9200 IF A$="" THEN A$=
```

```
     "FOR con valore iniziale errato":GOTO 1850
9210 BA$(LB)=BA$(LB)+A$+" TO "
9220 GOSUB 1940
9230 ME$=""
9240 FOR T=1 TO 6
9250 GOSUB 1770
9260 ME$=ME$+A$
9270 NEXT T
9280 MM=0
9290 IF LEFT$(ME$,2)="TO" THEN MM=2:ME$="TO"
9300 IF ME$="DOWNTO" THEN MM=6
9310 IF MM=0 THEN A$=
     "manca TO o DOWNTO dopo FOR":GOTO 1850
9320 GOSUB 2000
9330 FOR T=1 TO MM
9340 GOSUB 1770
9350 NEXT T
9360 GOSUB 4970
9370 BA$(LB)=BA$(LB)+A$
9380 IF A$="" THEN A$=
     "manca espressione dopo FOR":GOTO 1850
9390 GOSUB 1770
9400 MM$=A$
9410 GOSUB 1770
9420 MM$=MM$+A$
9430 IF MM$<>"DO" THEN A$="manca DO dopo
     FOR":GOTO1850
9440 IF ME$="DOWNTO" THEN BA$(LB)=BA$(LB)+"STEP-1"
9450 LB=LB+1
9460 RETURN
9470 '------------------------------------------------------------
```

Questo e' il risultato della compilazione
di un programma Pascal
```
***************************************************

<<<<< PASCAL >>>>>

CONST P=3.14;
VAR   A,B,C:REAL; I:INTEGER; K:CHAR; SW:BOOLEAN;

BEGIN

  READLN(B);
  FOR I:=15 DOWNTO 10 DO
    BEGIN
    C:=(B*P)
    WRITELN('circ=',C)
    END;

  WHILE K<>' ' DO
    BEGIN
    READLN(K)
    END;

  IF (B<C)OR(C=0) THEN
    BEGIN
    WRITE('//////////////')
    END
  ELSE
    BEGIN
    WRITE('%%%%%%%%%%%%')
    END;

 A:=ABS(B)+ATN(B)-COS(B)*EXP(B)/(-3.14E-2*B);
 A:=ROUND(B)+SIN(B)-SQR(B)*SQRT(B)-LN(B);
 A:=TRUNC(B)+ODD(I)-ORD(B)*PRED(B);
 K:=CHR(SUCC(22));A:=B MOD C +B DIV C;
 SW:=NOT FALSE;
 SW:=NOT(1 OR SW AND TRUE) OR (NOT FALSE);
 WRITELN('--------- FINE TEST ---------')

END.
```

BASIC

```
10   DEF FNOD(X)=((X-INT(X/2)*2)=1)
20   DEF FNPR(X)=X-1
30   DEF FNSU(X)=X+1
40   DEF FNRO(X)=INT(X+0.5)
50   DEF FNSQ(X)=X*X
60   REM ---CONST---
70   TRUE=-1
80   FALSE=0
90   P=3.314
100  REM ---BLOCK---
110  INPUT B
120  REM FOR
130  FOR I= 15 TO  10 STEP -1
140  REM BEGIN
150  C=( B *  P )
160  PRINT  "circ=", C
170  NEXT :REM END
180  REM WHILE
190  IF NOT ( K$<> " ") THEN GOTO  230
200  REM BEGIN
210  INPUT K$
220  GOTO  180:REM END
230  IF ( B< C) OR ( C= 0) THEN  260
240  IF NOT(( B< C) OR ( C= 0)) THEN  300
250  REM IF
260  REM THEN
270  REM BEGIN
280  PRINT  "/////////////";
290  GOTO  330:REM END
300  REM ELSE
310  REM BEGIN
320  PRINT  "%%%%%%%%%%%%%";
330  REM END
340  A=ABS( B)+ATN( B)-COS( B) * EXP( B)
     / (- 3.314E-2 *  B )
350  A=FNRO( B)+SIN( B)-FNSQ( B) * SQR( B) -LN( B)
360  A=INT( B)+FNOD( 1)-ASC( B) * FNPR( B)
370  K$=CHR$(FNSU( 22))
380  A= B MOD  C + B DIV  C
```

```
390  SW=NOT FALSE
400  SW=NOT( 1 OR  SW AND  TRUE ) OR (NOT FALSE)
410  PRINT  "---------- FINE TEST ----------"
420  END
```

COMPILATORE PASCAL PER SPECTRUM

```
  1 REM __ Programma ESEMPIO __
  2 REM
 10 DATA "CONST PI=3.14159;"
 20 DATA "VAR A,C,R:REAL;"
 30 DATA "     K,CONT:INTEGER;"
 40 DATA "     CH:CHAR;"
 50 DATA "     SW:BOOLEAN;"
 60 DATA " "
100 DATA "BEGIN"
105 DATA " "
110 DATA " CH:='Y';"
115 DATA " "
120 DATA " WHILE CH='Y' DO"
125 DATA " "
130 DATA "  BEGIN"
140 DATA "   FOR CONT:=1 TO 5 DO"
150 DATA "    BEGIN"
160 DATA "     WRITELN('raggio = ');"
170 DATA "     READLN(R);"
180 DATA "     C:=2*R*PI;"
190 DATA "     WRITELN('circonf.= ',C)"
200 DATA "    END;"
210 DATA " "
220 DATA "   WRITE('fine ? ');"
230 DATA "   READLN(CH)"
235 DATA " "
240 DATA "  END;"
250 DATA " "
270 DATA " A:=1.2E2;"
280 DATA " "
290 DATA " REPEAT"
300 DATA "  BEGIN"
310 DATA "   A:=A-1.5E-1"
320 DATA "  END"
330 DATA " UNTIL A<=0;"
340 DATA " "
400 DATA " IF (NOT TRUE = FALSE) THEN"
410 DATA "  BEGIN"
```

```
420 DATA "  SW:=TRUE AND FALSE OR TRUE"
430 DATA "  END"
440 DATA " ELSE"
450 DATA "  BEGIN"
460 DATA "  SW:= TRUE"
470 DATA "  END;"
480 DATA " "
500 DATA " A:=ABS(A)+ATN(A)-COS(A)/EXP(A)*ROUND(A)
    +SIN(A);"
510 DATA " A:=TRUNC(A)+ODD(5)-ORD(A)+PRED(5);"
520 DATA " A:=LN(A)+SQR(5E2)-SQRT(A)+SUCC(5);"
530 DATA " CH:=CHR(66);"
800 DATA " WRITELN ('*****FINE TEST******')"
900 DATA "END."
1000 REM
1010 REM
1020 REM PASCAL
1030 REM
1040 REM COMPILER
1050 REM
1060 REM
1070 REM
1080 REM BY JEREMY RUSTON
1090 REM COPYRIGHT (C) 1982
1100 REM
1110 REM
1120 REM
1130 REM
1140 GO SUB 1310
1150 REM
1160 DIM Y$(30): DIM B$(LE*2+8.50): DIM V$(20,30):
     DIM T(20):DIM C$(20,30):DIM C(20):DIM S$(50,50)
1170 LET SP=50
1180 LET LP=1
1190 LET CP=1
1200 LET PV=1
1210 LET PC=1
1220 LET D$=""
1230 LET E$=""
1240 LET F$=""
1250 LET G$=""
```

```
1260 REM
1270 GO SUB 5170
1280 REM
1290 GO SUB 1500
1300 STOP
1310 REM
1320 REM GET PASCAL
1330 REM
1340 RESTORE
1350 PRINT "PASCAL:"'
1360 LET LE=1
1370 READ A$
1380 PRINT A$
1390 IF A$="END." THEN  GO TO 1420
1400 LET LE=LE+1
1410 GO TO 1370
1420 LET LE=LE+1
1430 DIM P$(LE,50)
1440 RESTORE
1450 FOR T=1 TO LE-1
1460 READ P$(T)
1470 NEXT T
1480 LET P$(LE)="...................."
1490 RETURN
1500 REM
1510 REM OUTPUT BASIC
1520 REM
1530 PRINT ''''"BASIC:"'
1540 FOR T=1 TO LB-1
1545 LET Z$=B$(T): GO SUB 9500
1550 PRINT T*10;" ";Z$
1560 NEXT T
1570 RETURN
1580 REM
1590 REM INCREMENT PASCAL POINTERS
1600 REM
1610 LET CP=CP+1
1620 IF CP>LEN (P$(LP)) THEN  LET CP=1: LET LP=LP+1
1630 RETURN
1640 REM
1650 REM DECREMENT PASCAL POINTERS
```

```
1660 REM
1670 LET CP=CP-1
1680 IF CP=0 THEN  LET LP=LP-1:LET CP=LEN (P$(LP))
1690 RETURN
1700 REM
1710 REM GET NEXT PASCAL CHARACTER
1720 REM AND INCREMENT POINTERS
1730 REM
1740 LET A$=P$(LP,CP)
1750 GO SUB 1580
1760 RETURN
1770 REM
1780 REM GET NEXT PASCAL CHARACTER
1790 REM IGNORING SPACES,AND
1800 REM INCREMENT POINTERS
1810 REM
1820 GO SUB 1700
1830 IF A$=" " THEN  GO TO 1820
1840 RETURN
1850 REM
1860 REM ERROR
1870 REM
1880 PRINT '''
1890 IF LP<>1 THEN  PRINT P$(LP-1)
1900 PRINT P$(LP)
1910 IF LP<>LE THEN  PRINT P$(LP+1)
1920 PRINT ">>>>>>ERROR -"'A$
1930 STOP
1940 REM
1950 REM SAVE PASCAL POINTERS
1960 REM
1970 LET LG=LP
1980 LET CG=CP
1990 RETURN
2000 REM
2010 REM RESTORE PASCAL POINTERS
2020 REM
2030 LET LP=LG
2040 LET CP=CG
2050 RETURN
2060 REM
```

```
2070 REM GET DIGIT
2080 REM
2090 GO SUB 1940
2100 GO SUB 1770
2110 IF A$>="0" AND A$<="9" THEN  RETURN
2120 GO SUB 2000
2130 LET A$=""
2140 RETURN
2150 REM
2160 REM GET LETTER
2170 REM
2180 GO SUB 1940
2190 GO SUB 1770
2200 IF A$>="A" AND A$<="Z" THEN  RETURN
2210 GO SUB 2000
2220 LET A$=""
2230 RETURN
2240 REM
2250 REM GET UNSIGNED INTEGER
2260 REM
2270 LET H$=""
2275 GO SUB 2060: GO TO 2290
2280 GO SUB 1940: GO SUB 1700: GO SUB 2110
2290 IF A$="" THEN  LET A$=H$: RETURN
2300 LET H$=H$+A$
2310 GO TO 2280
2320 REM
2330 REM GET IDENTIFIER
2340 REM
2350 LET I$=""
2360 GO SUB 2150
2370 IF A$="" THEN  RETURN
2380 LET I$=I$+A$
2390 GO SUB 1940: GO SUB 1700: GO SUB 2110
2400 LET J$=A$
2410 LET I$=I$+A$
2420 GO SUB 1940: GO SUB 1700: GO SUB 2200
2430 IF A$="" AND J$="" THEN  LET A$=I$: RETURN
2440 GO TO 2380
2450 REM
2460 REM GET UNSIGNED NUMBER
```

```
2470 REM
2480 LET K$=""
2490 GO SUB 2240
2500 IF A$="" THEN  RETURN
2510 LET K$=K$+A$
2520 GO SUB 1940
2530 GO SUB 1700
2540 GO SUB 2000
2550 IF A$<>"." THEN  GO TO 2620
2560 GO SUB 1770
2570 LET K$=K$+A$
2580 GO SUB 2240
2590 IF A$="" THEN  RETURN
2600 LET K$=K$+A$
2610 GO SUB 1940
2620 GO SUB 1700
2630 IF A$<>"E" THEN LET A$=K$: GO SUB 2000:RETURN
2640 LET K$=K$+A$
2650 GO SUB 1940
2660 GO SUB 1700
2670 GO SUB 2000
2680 IF A$="+" OR A$="-" THEN  LET K$=K$+A$:
     GO SUB 1700
2690 GO SUB 2240
2700 IF A$="" THEN  RETURN
2710 LET A$=K$+A$
2720 RETURN
2730 REM
2740 REM GET STRING
2750 REM
2760 GO SUB 1940
2770 GO SUB 1770
2780 GO SUB 2000
2790 IF A$<>"'" THEN  LET A$="": RETURN
2800 LET L$=CHR$ (34)
2810 GO SUB 1770
2820 GO SUB 1700
2830 LET L$=L$+A$
2840 IF A$<>"'" THEN  GO TO 2820
2850 LET A$=L$( TO LEN (L$)-1)+CHR$ (34)
2860 RETURN
```

```
2870 REM
2880 REM GET UNSIGNED CONSTANT
2890 REM
2900 LET CD=CP: LET LD=LP
2910 GO SUB 2320
2920 IF A$="" THEN  LET CP=CD: LET LP=LD:
     GO TO 3000
2930 LET FL=0
2940 FOR E=1 TO PC-1
2950 LET Y$=A$: IF C$(E)=Y$ THEN  LET FL=E
2960 NEXT E
2970 IF FL=0 THEN  LET A$="": LET CP=CD:
     LET LP=LD: RETURN
2980 IF C(FL)=4 THEN  LET A$=A$+"$"
2990 RETURN
3000 GO SUB 2450
3010 IF A$<>"" THEN  RETURN
3020 GO SUB 2730
3030 IF A$<>"" THEN  RETURN
3040 RETURN
3050 REM
3060 REM GET CONSTANT
3070 REM
3080 GO SUB 2320
3090 IF A$="" THEN  GO TO 3170
3100 LET FL=0
3110 FOR F=1 TO PC-1
3120 LET Y$=A$: IF C$(F)=Y$ THEN  LET FL=F
3130 NEXT F
3140 IF FL=0 THEN  LET A$="": RETURN
3150 IF C(FL)=4 THEN  LET A$=A$+"$"
3160 RETURN
3170 LET M$=""
3180 GO SUB 1940
3190 GO SUB 1770
3200 GO SUB 2000
3210 IF A$="+" OR A$="-" THEN  LET M$=A$:
     GO SUB 1770
3220 GO SUB 2450
3230 IF A$<>"" THEN  LET A$=M$+A$: RETURN
3240 GO SUB 2730
```

```
3250 IF A$<>"" THEN  LET A$=M$+A$: RETURN
3260 LET A$="BAD CONSTANT IDENTIFIER"
3270 GO TO 1850
3280 REM
3290 REM GET VARIABLE
3300 REM
3310 LET LD=LP: LET CD=CP
3320 GO SUB 2320
3330 IF A$="" THEN  LET LP=LD: LET CP=CD: RETURN
3340 LET FL=0
3350 FOR G=1 TO PV-1
3360 LET Y$=A$: IF V$(G)=Y$ THEN  LET FL=G
3370 NEXT G
3380 IF FL>0 THEN  IF T(FL)=4 THEN  LET A$=A$+"$"
3390 IF FL<>0 THEN  RETURN
3400 LET A$=""
3410 LET LP=LD: LET CP=CD
3420 RETURN
3430 REM
3440 REM PUSH SOME VARS
3450 REM
3460 LET S$(SP)=D$
3470 LET S$(SP-1)=E$
3480 LET S$(SP-2)=F$
3490 LET S$(SP-3)=G$
3500 LET SP=SP-4
3510 RETURN
3520 REM
3530 REM PUSH A$ TO STACK
3540 REM
3550 LET S$(SP)=A$
3560 LET SP=SP-1
3570 RETURN
3580 REM
3590 REM PULL SOME VARS
3600 REM
3610 LET G$=S$(SP+1)
3620 LET F$=S$(SP+2)
3630 LET E$=S$(SP+3)
3640 LET D$=S$(SP+4)
3650 LET SP=SP+4
```

```
3651 IF G$>"" THEN  IF G$(LEN G$)=" " THEN  LET
     G$=G$( TO LEN G$-1): GO TO 3651
3652 IF F$>"" THEN  IF F$(LEN F$)=" " THEN  LET
     F$=F$( TO LEN F$-1): GO TO 3652
3653 IF E$>"" THEN  IF E$(LEN E$)=" " THEN  LET
     E$=E$( TO LEN E$-1): GO TO 3653
3654 IF D$>"" THEN  IF D$(LEN D$)=" " THEN  LET
     D$=D$( TO LEN D$-1): GO TO 3654
3660 RETURN
3670 REM
3680 REM PULL A$ FROM STACK
3690 REM
3700 LET SP=SP+1
3710 LET A$=S$(SP)
3711 IF A$>"" THEN  IF A$(LEN A$)=" " THEN  LET
     A$=A$( TO LEN A$-1): GO TO 3711
3720 RETURN
3730 REM
3740 REM GET FACTOR
3750 REM
3760 GO SUB 1940
3770 LET N$=""
3780 FOR G=1 TO 3
3790 GO SUB 1770
3800 LET N$=N$+A$
3810 NEXT G
3830 IF N$<>"NOT" THEN  LET N$="": GO SUB 2000
3840 GO SUB 2870
3850 IF A$<>"" THEN  LET A$=N$+" "+A$: RETURN
3860 GO SUB 3280
3870 IF A$<>"" THEN  LET A$=N$+" "+A$: RETURN
3880 GO SUB 1940
3890 GO SUB 1770
3900 GO SUB 2000
3910 IF A$<>"(" THEN  GO TO 4040
3920 GO SUB 1770
3930 LET A$=N$+"("
3940 GO SUB 3520
3950 GO SUB 3430
3960 GO SUB 4970
3970 GO SUB 3580
```

```
3980 LET N$=A$
3990 GO SUB 1770
4000 IF A$<>")" THEN  LET A$="NO CLOSING BRACKET":
     GO TO 1850
4010 GO SUB 3670
4020 LET A$=A$+N$+")"
4030 RETURN
4040 GO SUB 1940
4050 LET O$=N$
4060 LET N$=""
4070 FOR G=1 TO 5
4080 GO SUB 1770
4090 LET N$=N$+A$
4100 NEXT G
4110 GO SUB 2000
4120 LET GN=0
4130 IF N$(TO 3)="ABS"THEN LET GN=3: LET A$="ABS"
4140 IF N$(TO 3)="ATN"THEN LET GN=3: LET A$="ATN"
4150 IF N$(TO 3)="CHR"THEN LET GN=3: LET A$="CHR$"
4160 IF N$(TO 3)="COS"THEN LET GN=3: LET A$="COS"
4170 IF N$(TO 3)="EXP"THEN LET GN=3: LET A$="EXP"
4180 IF N$(TO 2)="LN"THEN LET GN=2: LET A$="LN"
4190 IF N$(TO 3)="ODD"THEN LET GN=3: LET A$="FN OD"
4200 IF N$( TO 3)="ORD" THEN  LET GN=3:
     LET A$="CODE"
4210 IF N$( TO 4)="PRED" THEN  LET GN=4:
     LET A$="FN PR"
4220 IF N$( TO 5)="ROUND" THEN  LET GN=5:
     LET A$="FN RO"
4230 IF N$( TO 3)="SIN" THEN  LET GN=3:
     LET A$="SIN"
4240 IF N$( TO 4)="SQRT" THEN  LET GN=4:
     LET A$="SQR": GO TO 4280
4250 IF N$( TO 3)="SQR" THEN  LET GN=3:
     LET A$="FN SQ"
4260 IF N$( TO 4)="SUCC" THEN  LET GN=4:
     LET A$="FN SU"
4270 IF N$( TO 5)="TRUNC" THEN  LET GN=5:
     LET A$="INT"
4280 IF GN=0 THEN  LET A$="BAD FUNTION CALL":
     GO TO 1850
```

```
4290 LET O$=O$+A$
4300 FOR G=1 TO GN
4310 GO SUB 1770
4320 NEXT G
4330 GO SUB 1770
4340 IF A$<>"(" THEN  LET A$="NO OPENING BRACKET
     FOR FUNCTION": GO TO 1850
4350 LET O$=O$+A$
4360 LET A$=O$
4370 GO SUB 3520
4380 GO SUB 3430
4390 GO SUB 4970
4400 GO SUB 3580
4410 LET O$=A$
4420 GO SUB 3670
4430 LET O$=A$+O$
4440 GO SUB 1770
4450 IF A$<>")" THEN  LET A$="NO CLOSING BRACKET
     FOR FUNCTION CALL": GO TO 1850
4460 LET A$=O$+")"
4470 RETURN
4480 REM
4490 REM GET TERM
4500 REM
4510 GO SUB 3730
4520 IF A$="" THEN  LET A$="MISSING FACTOR":
     GO TO 1850
4530 LET D$=A$
4540 LET E$=""
4550 GO SUB 1940
4560 FOR J=1 TO 3
4570 GO SUB 1770
4580 LET E$=E$+A$
4590 NEXT J
4600 GO SUB 2000
4610 IF E$<>"DIV" AND E$<>"MOD" AND E$<>"AND"
     AND E$(1)<>"*" AND E$(1)<>"/" THEN LET A$=D$:
     RETURN
4620 LET JA=1
4630 IF E$="AND" OR E$="DIV" OR E$="MOD" THEN
     LET JA=3
```

```
4640 FOR T=1 TO JA
4650 GO SUB 1770
4660 NEXT T
4670 GO SUB 3730
4680 IF A$="" THEN  LET A$="MISSING FACTOR": GO
     TO 1850
4690 LET D$=D$+" "+E$( TO JA)+" "+A$+" "
4700 GO TO 4540
4710 REM
4720 REM SIMPLE EXPRESSION
4730 REM
4740 GO SUB 1940
4750 GO SUB 1770
4760 GO SUB 2000
4770 LET F$=""
4780 IF A$="+" OR A$="-" THEN  LET F$=A$:
     GO SUB 1770
4790 GO SUB 4480
4800 IF A$="" THEN  LET A$="MISSING TERM":
     GO TO 1850
4810 LET F$=F$+A$
4820 GO SUB 1940
4830 LET Q$=""
4840 GO SUB 1770
4850 LET Q$=Q$+A$
4860 GO SUB 1770
4870 LET Q$=Q$+A$
4880 GO SUB 2000
4890 IF Q$(1)<>"+" AND Q$(1)<>"-" AND Q$<>"OR"
     THEN  LET A$=F$: RETURN
4900 IF Q$<>"OR" THEN  LET F$=F$+Q$(1)
4910 GO SUB 1770
4920 IF Q$="OR" THEN  LET F$=F$+" OR ":GO SUB 1770
4930 GO SUB 4480
4940 LET F$=F$+A$
4950 IF A$="" THEN  LET A$=F$: RETURN
4960 GO TO 4820
4970 REM
4980 REM GET EXPRESSION
4990 REM
5000 GO SUB 4710
```

```
5010 LET G$=A$
5020 GO SUB 1940
5030 LET R$=""
5040 FOR M=1 TO 2
5050 GO SUB 1770
5060 LET R$=R$+A$
5070 NEXT M
5080 GO SUB 2000
5090 LET T$=R$(1)
5100 IF T$<>"<" AND T$<>"=" AND T$<>">" AND T$<>"
     <=" AND T$<>"<>" AND T$<>">=" THEN  LET A$=
     G$: RETURN
5110 IF R$="<>" OR R$="<=" OR R$=">=" THEN  GO SUB
     1770: GO SUB 1770: LET A$=R$:GO TO 5130
5120 GO SUB 1770: LET A$=T$
5130 LET G$=G$+A$
5140 GO SUB 4710
5150 LET A$=G$+A$
5160 RETURN
5170 REM
5180 REM
5190 REM
5200 REM
5210 REM
5220 REM
5230 REM
5240 REM MAIN PROGRAM
5250 REM
5260 REM
5270 REM
5280 REM
5290 REM
5300 LET B$(1)="DEF FN OD(X)=((X-INT (X/2)*2)=1)"
5310 LET B$(2)="DEF FN PR(X)=X-1"
5320 LET B$(3)="DEF FN SU(X)=X+1"
5330 LET B$(4)="DEF FN RO(X)=INT(X+0.5)"
5340 LET B$(5)="DEF FN SQ(X)=X*X"
5350 LET LB=6
5360 GO SUB 5400
5370 GO SUB 5790
5380 GO SUB 6230
```

```
5390 RETURN
5400 REM
5410 REM CONST SECTION
5420 REM
5430 LET C$(1)="TRUE": LET C$(2)="FALSE": LET PC=3
5440 LET B$(LB)="REM ---CONST---"
5450 LET B$(LB+1)="LET TRUE=1"
5460 LET B$(LB+2)="LET FALSE=0"
5470 LET LB=LB+3
5480 LET U$=""
5490 GO SUB 1940
5500 FOR M=1 TO 5
5510 GO SUB 1770
5520 LET U$=U$+A$
5530 NEXT M
5540 IF U$<>"CONST" THEN  GO SUB 2000: RETURN
5550 GO SUB 2320
5560 IF A$="" THEN  LET A$="MISSING CONSTANT
     IDENTIFIER": GO TO 1850
5570 LET C$(PC)=A$
5580 GO SUB 1770
5590 IF A$<>"=" THEN  LET A$="MISSING EQUALS SIGN":
     GO TO 1850
5600 GO SUB 3050
5610 IF A$="" THEN  LET A$="MISSING CONSTANT":
     GO TO 1850
5620 IF A$(1)=CHR$ (34) THEN  LET T(PC)=4
5630 LET B$(LB)=C$(PC)
5640 IF T(PC)=4 THEN  LET Z$=B$(LB): GO SUB 9500:
     LET B$(LB)=Z$+"$"
5650 LET Z$=B$(LB): GO SUB 9500: LET B$(LB)="LET
     "+Z$+"="+A$
5660 LET LB=LB+1
5670 LET PC=PC+1
5680 GO SUB 1770
5690 IF A$<>";" THEN  LET A$="MISSING SEMI COLON"
     : GO TO 1850
5700 GO SUB 1940
5710 LET U$=""
5720 FOR M=1 TO 5
5730 GO SUB 1770
```

```
5740 LET U$=U$+A$
5750 NEXT M
5760 GO SUB 2000
5770 IF U$="BEGIN" OR U$( TO 3)="VAR" THEN  RETURN
5780 GO TO 5550
5790 REM
5800 REM VAR SECTION
5810 REM
5820 GO SUB 1940
5830 LET U$=""
5840 FOR T=1 TO 3
5850 GO SUB 1770
5860 LET U$=U$+A$
5870 NEXT T
5880 GO SUB 2000
5890 IF U$<>"VAR" THEN  RETURN
5900 GO SUB 1770: GO SUB 1770: GO SUB 1770
5910 LET A$="END"
5920 GO SUB 3520
5930 GO SUB 2320
5940 IF A$="" THEN  LET A$="MISSING IDENTIFIER":
     GO TO 1850
5950 GO SUB 3520
5960 GO SUB 1770
5970 IF A$=":" THEN  GO TO 6000
5980 IF A$<>"," THEN  LET A$="MISSING COMMA
     IN VAR SECTION": GO TO 1850
5990 GO TO 5930
6000 GO SUB 2320
6010 IF A$<>"REAL" AND A$<>"INTEGER" AND A$<>"
     CHAR" AND A$<>"BOOLEAN"THEN  LET A$="ILLEGAL
     TYPE": GO TO 1850
6020 IF A$="REAL" THEN  LET MM=1
6030 IF A$="INTEGER" THEN  LET MM=2
6040 IF A$="BOOLEAN" THEN  LET MM=3
6050 IF A$="CHAR" THEN  LET MM=4
6060 GO SUB 1770
6070 IF A$<>";" THEN  LET A$="MISSING SEMI COLON":
     GO TO 1850
6080 GO SUB 3670
6090 IF A$="END" THEN  GO TO 6140
```

```
6100 LET V$(PV)=A$
6110 LET T(PV)=MM
6120 LET PV=PV+1
6130 GO TO 6080
6140 LET U$=""
6150 GO SUB 1940
6160 FOR T=1 TO 5
6170 GO SUB 1770
6180 LET U$=U$+A$
6190 NEXT T
6200 GO SUB 2000
6210 IF U$="BEGIN" THEN  RETURN
6220 GO TO 5910
6230 REM
6240 REM BLOCK
6250 REM
6260 LET B$(LB)="REM ---BLOCK---"
6270 LET LB=LB+1
6280 LET U$=""
6290 LET EN=0
6300 FOR T=1 TO 5
6310 GO SUB 1770
6320 LET U$=U$+A$
6330 NEXT T
6340 IF U$<>"BEGIN" THEN  LET A$="MISSING BEGIN":
     GO TO 1850
6350 GO SUB 7090
6360 IF EN=0 THEN  GO TO 6350
6370 REM
6380 REM START SECOND PASS
6390 REM
6400 FOR T=1 TO LB-1
6410 LET A$=B$(T)
6420 IF A$(TO 13)="REM ---END---" THEN GO SUB 6460
6430 IF A$( TO 15)="REM ---UNTIL---" THEN  GO
     SUB 6970
6440 NEXT T
6450 RETURN
6460 LET LI=T-1
6470 LET BE=0
6480 IF B$(LI, TO 15)="REM ---BEGIN---" AND BE=0
```

```
     THEN  GO TO 6540
6490 IF B$(LI, TO 15)="REM ---BEGIN---" THEN
     LET BE=BE+1
6500 IF B$(LI, TO 13)="REM ---END---" THEN
     LET BE=BE-1
6510 LET LI=LI-1
6520 IF LI=0 THEN  PRINT "CAN'T MATCH BEGIN/END":
     GO TO 1850
6530 GO TO 6480
6540 IF B$(LI-2, TO 12)="REM ---IF---" THEN
     GO TO 6750
6550 IF B$(LI-2, TO 15)="REM ---WHILE---" THEN
     GO TO 6700
6560 IF B$(LI-2, TO 13)="REM ---FOR---" THEN
     GO TO 6600
6570 LET B$(LI)="REM BEGIN"
6580 LET B$(T)="REM END"
6590 RETURN
6600 LET B$(LI)="REM BEGIN"
6610 LET B$(LI-2)="REM FOR"
6620 LET U$=""
6630 LET H=5
6640 LET U$=U$+B$(LI-1,H)
6650 LET H=H+1
6660 IF B$(LI-1,H)="=" THEN  GO TO 6680
6670 GO TO 6640
6680 LET B$(T)="NEXT "+U$+" :REM END"
6690 RETURN
6700 LET B$(LI)="REM BEGIN"
6710 LET B$(LI-2)="REM WHILE"
6720 LET B$(T)="GO TO "+STR$ ((LI-2)*10)+":REM END"
6730 LET Z$=B$(LI-1): GO SUB 9500: LET B$(LI-1)
     =Z$+" "+STR$ ((T+1)*10)
6740 RETURN
6750 LET EL=0
6760 IF B$(T+1,TO 14)="REM ---ELSE---" THEN LET EL=1
6770 LET B$(LI-1)="REM THEN"
6780 LET Z$=B$(LI-3): GO SUB 9500: LET B$(LI-3)
     =Z$+" "+STR$ ((T+1)*10)
6790 LET B$(LI-2)="REM IF"
6800 LET Z$=B$(LI-4): GO SUB 9500: LET B$(LI-4)
```

```
     =Z$+" "+STR$ ((LI-1)*10)
6810 LET B$(LI)="REM BEGIN"
6820 LET B$(T)="REM END"
6830 IF EL=0 THEN  RETURN
6840 LET B$(T+1)="REM ELSE"
6850 LET B$(T+2)="REM BEGIN"
6860 LET EN=0
6870 LET LL=T+3
6880 IF B$(LL, TO 13)="REM ---END---" AND EN=0
     THEN  GO TO 6940
6890 IF B$(LL, TO 13)="REM ---END---" THEN
     LET EN=EN+1
6900 IF B$(LL, TO 15)="REM ---BEGIN---" THEN
     LET EN=EN-1
6910 LET LL=LL+1
6920 IF LL>=LB THEN  PRINT "ELSE SECTION BAD":STOP
6930 GO TO 6880
6940 LET B$(T)="GOTO "+STR$ (LL*10)+":REM END"
6950 LET B$(LL)="REM END"
6960 RETURN
6970 LET LI=T-1
6980 LET RE=0
6990 IF B$(LI, TO 16)="REM ---REPEAT---" AND RE=0
     THEN  GO TO 7050
7000 IF B$(LI, TO 16)="REM ---REPEAT---" THEN
     LET RE=RE+1
7010 IF B$(LI. TO 15)="REM ---UNTIL---" THEN
     LET RE=RE-1
7020 LET LI=LI-1
7030 IF LI=0 THEN  PRINT "CAN'T MATCH REPEAT/UNTIL
     ": STOP
7040 GO TO 6990
7050 LET Z$=B$(T+1): GO SUB 9500: LET B$(T+1)=Z$+"
     "+STR$ (LI*10)
7060 LET B$(LI)="REM REPEAT"
7070 LET B$(T)="REM UNTIL"
7080 RETURN
7090 REM
7100 REM GET STATEMENT
7110 REM
7120 LET U$=""
```

```
7130 GO SUB 1940
7140 FOR T=1 TO 6
7150 GO SUB 1770
7160 LET U$=U$+A$
7170 NEXT T
7180 GO SUB 2000
7190 IF U$( TO 5)="WRITE" THEN  GO TO 7480
7200 IF U$="READLN" THEN  GO TO 7780
7210 IF U$( TO 5)="BEGIN" THEN  GO TO 8000
7220 IF U$( TO 3)="END" THEN  GO TO 8090
7230 IF U$( TO 2)="IF" THEN  GO TO 8260
7240 IF U$( TO 4)="THEN" THEN  GO TO 8460
7250 IF U$( TO 4)="ELSE" THEN  GO TO 8550
7260 IF U$="REPEAT" THEN  GO TO 8640
7270 IF U$( TO 5)="UNTIL" THEN  GO TO 8730
7280 IF U$( TO 5)="WHILE" THEN  GO TO 8860
7290 IF U$( TO 3)="FOR" THEN  GO TO 9040
7300 REM
7310 REM ASSIGNAMENT
7320 REM
7330 GO SUB 3280
7340 IF A$="" THEN  LET A$="INVALID VARIABLE":
     GO TO 1850
7350 LET B$(LB)="LET "+A$+"="
7360 GO SUB 1770
7370 IF A$<>":" THEN  LET A$="MISSING COLON":
     GO TO 1850
7380 GO SUB 1770
7390 IF A$<>"=" THEN  LET A$="MISSING EQUALS SIGN"
     : GO TO 1850
7400 GO SUB 4970
7410 IF A$="" THEN  LET A$="NO EXPRESSION":
     GO TO 1850
7420 LET Z$=B$(LB): GO SUB 9500: LET B$(LB)=Z$+A$
7430 GO SUB 1940
7440 GO SUB 1770
7450 IF A$<>";" THEN  GO SUB 2000
7460 LET LB=LB+1
7470 RETURN
7480 REM
7490 REM WRITE/WRITELN
```

```
7500 REM
7510 FOR T=1 TO 5
7520 GO SUB 1770
7530 NEXT T
7540 GO SUB 1940
7550 GO SUB 1770
7560 LET U$=A$
7570 GO SUB 1770
7580 LET U$=U$+A$
7590 IF U$<>"LN" THEN  LET U$="": GO SUB 2000
7600 LET B$(LB)="PRINT"
7610 GO SUB 1770
7620 IF A$<>"(" THEN  LET A$="MISSING OPENING
     BRACKET": GO TO 1850
7630 GO SUB 1940
7640 GO SUB 1770
7650 GO SUB 2000
7660 IF A$=")" THEN  GO TO 1770: GO TO 7720
7670 GO SUB 4970
7680 LET Z$=B$(LB): GO SUB 9500: LET B$(LB)=Z$+A$
7690 GO SUB 1770
7700 IF A$="," THEN  LET Z$=B$(LB): GO SUB 9500:
      LET B$(LB)=Z$+A$: GO TO 7630
7710 IF A$<>")" THEN  LET A$="ILLEGAL SEPARATOR":
     GO TO 1850
7720 IF U$<>"LN" THEN  LET Z$=B$(LB): GO SUB 9500:
     LET B$(LB)=Z$+";"
7730 LET LB=LB+1
7740 GO SUB 1940
7750 GO SUB 1770
7760 IF A$<>";" THEN  GO SUB 2000
7770 RETURN
7780 REM
7790 REM READLN
7800 REM
7810 FOR T=1 TO 6
7820 GO SUB 1770
7830 NEXT T
7840 GO SUB 1770
7850 IF A$<>"(" THEN  LET A$="MISSING OPENING
     BRACKET": GO TO 1850
```

```
7860 LET B$(LB)="INPUT "
7870 GO SUB 3280
7880 IF A$="" THEN  LET A$="MISSING VARIABLE":
     GO TO 1850
7890 LET Z$=B$(LB):GO SUB 9500: LET B$(LB)=Z$+""+A$
7900 GO SUB 1770
7910 IF A$=")" THEN  GO TO 7950
7920 IF A$<>"," THEN  LET A$="MISSING COMMA":
     GO TO 1850
7930 LET Z$=B$(LB): GO SUB 9500: LET B$(LB)=Z$+","
7940 GO TO 7870
7950 GO SUB 1940
7960 GO SUB 1770
7970 IF A$<>";" THEN  GO SUB 2000
7980 LET LB=LB+1
7990 RETURN
8000 REM
8010 REM BEGIN
8020 REM
8030 LET B$(LB)="REM ---BEGIN---"
8040 LET LB=LB+1
8050 FOR T=1 TO 5
8060 GO SUB 1770
8070 NEXT T
8080 RETURN
8090 REM
8100 REM END
8110 REM
8120 FOR T=1 TO 3
8130 GO SUB 1770
8140 NEXT T
8150 GO SUB 1940
8160 GO SUB 1770: IF A$=";" THEN  GO TO 8230
8170 GO SUB 2000
8180 IF A$<>"." THEN  GO TO 8230
8190 LET EN=1
8200 LET B$(LB)="STOP"
8210 LET LB=LB+1
8220 RETURN
8230 LET B$(LB)="REM ---END---"
8240 LET LB=LB+1
```

```
8250 RETURN
8260 REM
8270 REM IF
8280 REM
8290 GO SUB 1770: GO SUB 1770
8300 GO SUB 4970
8310 IF A$=""THEN LET A$="NO CONDITION":GO TO 1850
8320 LET B$(LB)="IF "+A$+" THEN GOTO"
8330 LET LB=LB+1
8340 LET B$(LB+1)="REM ---IF---"
8350 LET B$(LB)="IF NOT "+A$+" THEN GOTO"
8360 LET LB=LB+2
8370 LET U$=""
8380 GO SUB 1940
8390 FOR T=1 TO 4
8400 GO SUB 1770
8410 LET U$=U$+A$
8420 NEXT T
8430 IF U$<>"THEN" THEN  LET A$="MISSING THEN":
     GO TO 1850
8440 GO SUB 2000
8450 RETURN
8460 REM
8470 REM THEN
8480 REM
8490 FOR T=1 TO 4
8500 GO SUB 1770
8510 NEXT T
8520 LET B$(LB)="REM ---THEN---"
8530 LET LB=LB+1
8540 RETURN
8550 REM
8560 REM ELSE
8570 REM
8580 FOR T=1 TO 4
8590 GO SUB 1770
8600 NEXT T
8610 LET B$(LB)="REM ---ELSE---"
8620 LET LB=LB+1
8630 RETURN
8640 REM
```

```
8650 REM REPEAT
8660 REM
8670 FOR T=1 TO 6
8680 GO SUB 1770
8690 NEXT T
8700 LET B$(LB)="REM ---REPEAT---"
8710 LET LB=LB+1
8720 RETURN
8730 REM
8740 REM UNTIL
8750 REM
8760 LET B$(LB)="REM ---UNTIL---"
8770 LET LB=LB+1
8780 FOR T=1 TO 5
8790 GO SUB 1770
8800 NEXT T
8810 GO SUB 4970
8820 IF A$="" THEN  LET A$="MISSING EXPRESSION":
     GO TO 1850
8830 LET B$(LB)="IF NOT("+A$+") THEN GOTO
8835 GO TO 7430
8840 LET LB=LB+1
8850 RETURN
8860 REM
8870 REM WHILE
8880 REM
8890 LET B$(LB)="REM ---WHILE---"
8900 LET LB=LB+1
8910 FOR T=1 TO 5
8920 GO SUB 1770
8930 NEXT T
8940 GO SUB 4970
8950 IF A$="" THEN  LET A$="MISSING EXPRESSION":
     GO TO 1850
8960 LET B$(LB)="IF NOT("+A$+") THEN GOTO
8970 GO SUB 1770
8980 LET U$=A$
8990 GO SUB 1770
9000 LET U$=U$+A$
9010 IF U$<>"DO" THEN  LET A$="MISSING DO":
     GO TO 1850
```

```
9020 LET LB=LB+1
9030 RETURN
9040 REM
9050 REM FOR
9060 REM
9070 LET B$(LB)="REM ---FOR---"
9080 LET LB=LB+1
9090 FOR T=1 TO 3
9100 GO SUB 1770
9110 NEXT T
9120 GO SUB 3280
9130 IF A$="" OR FL=0 OR T(FL+(FL=0))=4 THEN  LET
     A$="BAD FOR VARIABLE": GO TO 1850
9140 LET B$(LB)="FOR "+A$+"="
9150 GO SUB 1770
9160 IF A$<>":" THEN  LET A$="MISSING COLON":
     GO TO 1850
9170 GO SUB 1770
9180 IF A$<>"=" THEN  LET A$="MISSING EQUALS
     SIGN": GO TO 1850
9190 GO SUB 4970
9200 IF A$="" THEN  LET A$="MISSING EXPRESSION":
     GO TO 1850
9210 LET Z$=B$(LB): GO SUB 9500: LET B$(LB)=Z$
     +A$+" TO "
9220 GO SUB 1940
9230 LET U$=""
9240 FOR T=1 TO 6
9250 GO SUB 1770
9260 LET U$=U$+A$
9270 NEXT T
9280 LET MM=0
9290 IF U$( TO 2)="TO" THEN  LET MM=2: LET U$="TO"
9300 IF U$="DOWNTO" THEN  LET MM=6
9310 IF MM=0 THEN  LET A$="MISSING TO/DOWNTO":
     GO TO 1850
9320 GO SUB 2000
9330 FOR T=1 TO MM
9340 GO SUB 1770
9350 NEXT T
9360 GO SUB 4970
```

```
9370 LET Z$=B$(LB): GO SUB 9500: LET B$(LB)=Z$+"
     "+A$
9380 IF A$="" THEN  LET A$="MISSING EXPRESSION":
     GO TO 1850
9390 GO SUB 1770
9400 LET W$=A$
9410 GO SUB 1770
9420 LET W$=W$+A$
9430 IF W$<>"DO" THEN  LET A$="MISSING DO":
     GO TO 1850
9440 IF U$="DOWNTO" THEN  LET Z$=B$(LB): GO
     SUB 9500: LET B$(LB)=Z$+" STEP -1"
9450 LET LB=LB+1
9460 RETURN
9470
9480
9490
9500 REM TAKE TRAILING SPACES OFF Z$
9510
9520 LET SPC=LEN Z$
9530 IF Z$(SPC)<>" " THEN  GO TO 9600
9540 LET SPC=SPC-1
9550 IF SPC<>0 THEN  GO TO 9530
9560 LET Z$="": RETURN
9600 LET Z$=Z$( TO SPC)
9610 RETURN
9999 SAVE "COMPILER"
```

Questo libro si rivolge a chi desidera conoscere il PASCAL ed apprenderne l'uso in modo semplice e lineare: è quindi adatto anche a chi è alle prime armi nel campo dell'informatica.
Nel libro sono riportati i listati di due programmi compilatori per tradurre le istruzioni PASCAL in BASIC: questo consente al lettore di provare direttamente programmi in PASCAL sul suo personal computer senza dover affrontare la spesa di un vero compilatore PASCAL.
Il primo compilatore è scritto in Basic MICROSOFT, quindi è adatto ai personal computer IBM PC; IBM compatibili, OLIVETTI M 10 - M 20 - M 21 - M 24, HP 150.
Il secondo è scritto in Basic SINCLAIR per lo ZX SPECTRUM ed è fornito su **cassetta software allegata al libro**.

Cod. 9800

L. 25.000
prezzo comprensivo di cassetta

IMPARIAMO IL PASCAL SUL NOSTRO COMPUTER
di JEREMY RUSTON
edizioni
JCE